Optimal Planning Modulo Theories

Francesco Leofante

A thesis submitted in fulfillment
of the requirements of the degree of
Doctor of Philosophy

Advisors:

Professor Erika Ábrahám, RWTH Aachen University, Germany

Professor Armando Tacchella, University of Genoa, Italy

RWTH Aachen University, Germany

and

University of Genoa, Italy

Acknowledgements

This thesis is the result of four years of challenging work, which would have been way more challenging without the help and support of many.

Let me start by acknowledging the scientific support which I received throughout this journey. First of all, I am very much indebted to my advisors and friends Erika and Armando for being an inspiration and a model during all of these years. I am indebted too to each of my other coauthors, whose work and collaboration has had a direct impact on this thesis: Tim Niemueller and Gerhard Lakemeyer from RWTH Aachen University and Enrico Giunchiglia, from the University of Genoa. I have also been lucky enough to share this journey with many great colleagues and friends, among which the AIMSLab people in Italy, the amazing THS group in Germany and many more. I would also like to acknowledge the positive and stimulating environment I found during my research stay at the Artificial Intelligence group at the Universitat Pompeu Fabra in Barcelona: thank you guys!

Even more important than scientific support, however, are love and friendship. I shall begin by thanking my parents, who have done so much for me over the years, and Marta for her love and encouragement. You guys were always there, despite my constant complaining about everything – for those who don't know me: trust me, I do complain a lot! All of these years have been extremely enjoyable also thanks to all of the friends with whom I have had the chance to share this journey: to all of you goes my gratitude.

Abstract

Planning for real-world applications requires algorithms and tools with the ability to handle the complexity such scenarios entail. However, meeting the needs of such applications poses substantial challenges, both representational and algorithmic. On the one hand, expressive languages are needed to build faithful models. On the other hand, efficient solving techniques that can support these languages need to be devised. A response to this challenge is underway, and the past few years witnessed a community effort towards more expressive languages, including decidable fragments of first-order theories.

In this work we focus on planning with arithmetic theories and propose *Optimal Planning Modulo Theories*, a framework that attempts to provide efficient means of dealing with such problems. Leveraging generic Optimization Modulo Theories (OMT) solvers, we first present domain-specific encodings for optimal planning in complex logistic domains. We then present a more general, domain-independent formulation that allows to extend OMT planning to a broader class of well-studied numeric problems in planning. To the best of our knowledge, this is the first time OMT procedures are employed in domain-independent planning.

Contents

Chapter 1

Introduction

1.1 Research area, motivations and goals

Citing Geffner and Bonet [GB13], planning can be defined as the *model-based approach to intelligent behavior*, where a model of the world and of possible actions to be performed is used to decide on a sequence of actions that brings the world to a desired state.

Different computational models of planning have been proposed and studied during the years [RN10]. The simplest model that has been extensively studied in Artificial Intelligence is the so-called *classical planning* model, in which a single agent is acting in a fully observable world where actions have deterministic effects and the objective is to achieve a certain goal performing said actions. A solution to a classical planning problem is a sequence of actions that maps the initial situation into some situation that satisfies the goal description; such a sequence is called a *plan*. A fundamental distinction in planning is between finding *any* plan that solves a problem and finding a plan with minimum cost. We refer to the former as *satisficing* planning, and to the latter as *optimal* planning. The focus of this thesis is on optimal planning.

Although simplistic, the classical planning model is broad enough to describe many real-world combinatorial problems. Despite being PSPACE-complete in the worst case [Byl94, BN95], great advances have been obtained by the planning community. This was made possible by two factors: *(i)* the development of a

standard modeling language, PDDL [McD00], used to model planning problems and *(ii)* the parallel development of efficient algorithmic solutions based, e.g., on heuristic search, symbolic search or propositional satisfiability [BG01, RW10, ER99, KS92].

Of particular interest for the development of this work are the latter approaches. Indeed, this thesis builds upon, and extends, the so-called *Planning as Satisfiability* framework, first proposed by Kautz and Selman in their seminal 1992 paper [KS92]. There, the authors showed for the first time that the classical planning problem could be solved by translating it into a propositional formula and checking its satisfiability. This idea was later improved in [KS96] where efficient techniques for propositional satisfiability, combined with efficient translations, were shown to be a competitive approach to classical planning. Further improvements were proposed in the subsequent years, most notably by Jussi Rintanen [RHN06, Rin09, Rin12], making Planning as SAT one of the most efficient approaches to classical planning.

Already in the early 2000's however, fully automated planning systems for classical planning such as FF [Hof01] were showing impressive results on the benchmark suite of the first International Planning Competitions. Many saw these results as an indication that it was time to move on to richer modeling languages than the propositional fragment of PDDL used to describe classical planning problems. As a result of this, PDDL2.1 [FL03] introduced several new, interesting features, among which, support for numeric variables and specification of more complex metrics for plan quality than mere plan length. Planning problems with *numeric features* and *optimization metrics* are indeed the focus of this thesis.

Despite early results showing undecidability of unrestricted numeric planning [Hel02], new approaches that could handle fragments of this added expressiveness started appearing. Examples include solutions based on heuristic search [HG01, DK01] and, notably, satisfiability-based approaches [WW99], [SD05]. Extremely relevant for this work is the latter approach, which extends the Planning as SAT framework to deal with real-valued quantities, among others. When describing the workings of their approach, the authors of [SD05] refer to a *SAT-based arithmetic constraint solver* that could handle Boolean combi-

nations of propositional and arithmetic constraints. These were the early days of Satisfiability Modulo Theories [BSST09] – when, in fact, this name was not yet popular.

Satisfiability Modulo Theories is concerned with checking the satisfiability of logical formulas over one or more theories. SMT draws on some the most fundamental problems of symbolic logic: the decision problem, completeness and incompleteness of logical theories, and finally complexity theory. The computational complexity of most SMT problems is typically very high, nevertheless efficient decision procedures have been developed and related tools have been implemented. One prominent example is the theory of quantifier-free linear arithmetic, for which efficient solvers exist nowadays [dMB08, CGSS13] and are routinely used in the area of Computer-Aided Verification.

This very theory has seen several applications in planning as well. Indeed, encouraged by the impressive progress in the field of SMT, several reductions from expressive planning models to SMT have been proposed in the last decade. To cite a few examples, Rintanen uses SMT to solve temporal planning problems [Rin15, Rin17], while Cashmore *et al.* [CFLM16] leverage SMT to plan in hybrid domains, i.e., domains featuring both discrete and continuous dynamics.

Is it the end of the story? Not really! Earlier in this introduction we said that this thesis is concerned with optimal planning in numeric domains featuring non-trivial optimization metrics. The reader may legitimately be wondering how such problems could be encoded as SMT formulas. Indeed, while SMT formulas can encode arbitrary arithmetic expressions – as long as the related theory remains decidable – it is not clear how to perform optimal reasoning.

Luckily for the author of this thesis, the SMT community has recognized the importance of finding optimal solutions to SMT formulas in recent years. As a result, standard procedures for SMT have been extended with optimization capabilities, leading to the development of a new, powerful framework: *Optimization Modulo Theories* (OMT) [ST15a]. OMT solvers such as [BPF15, ST15b] extend SMT solving with optimization procedures to find a variable assignment that defines an optimal value for a desired objective function under all models of a given SMT formula.

Having spoken about numeric planning with optimization metrics and OMT

solving, the objective of this thesis should now be clear – hopefully. The rest of this document will guide the reader through our attempts to use general OMT solvers to implement optimal numeric planners. Starting from a domain-dependent solution, we will later present what, to the best of our knowledge, is the first reduction from optimal numeric planning to OMT.

1.2 Thesis outline

The remainder of this document is organized as follows. Chapter 2 introduces most of the necessary background on the topics we touch in this thesis. Chapter 3 discusses and evaluates our efforts towards applying OMT to domain-specific planning for logistic domains. Chapter 4 presents the general Optimal Planning Modulo Theories framework for domain-independent planning. We discuss new encodings for planning problems that are more amenable to optimal planning as OMT, along with an experimental study with benchmarks for numeric planning taken from the literature. Chapter 5 presents the OMTPlan planner where all the ideas presented in Chapter 4 have been implemented. We conclude this thesis in Chapter 6 by summarizing our findings and outlining some possible directions for future research.

1.3 Relevant publications

In this section we list the articles that contain contributions presented in this thesis. Entries listed below, with the exception of the last one, represent the core on which Chapter 3 is based.

1.3.1 Peer-reviewed publications

- [NLLÁ17] Tim Niemueller, Gerhard Lakemeyer, Francesco Leofante, and Erika Ábrahám. Towards CLIPS-based task execution and monitoring with SMT-based decision optimization. In *Proc. of PlanRob@ICAPS*, pages 60–67, 2017

- [LÁN+17] Francesco Leofante, Erika Ábrahám, Tim Niemueller, Gerhard Lakemeyer, and Armando Tacchella. On the synthesis of guaranteed-

quality plans for robot fleets in logistics scenarios via optimization modulo theories. In *Proc. of IRI*, pages 403–410, 2017

- [LÁT18] Francesco Leofante, Erika Ábrahám, and Armando Tacchella. Task planning with OMT: an application to production logistics. In *Proc. of IFM*, pages 316–325, 2018

- [Leo18a] Francesco Leofante. Guaranteed plans for multi-robot systems via Optimization Modulo Theories. In *Proc. of AAAI*, pages 8020–8021, 2018

- [Leo18b] Francesco Leofante. Optimal multi-robot task planning: from synthesis to execution (and back). In *Proc. of IJCAI*, pages 5771–5772, 2018

- [LÁN+19] Francesco Leofante, Erika Ábrahám, Tim Niemueller, Gerhard Lakemeyer, and Armando Tacchella. Integrated synthesis and execution of optimal plans for multi-robot systems in logistics. *Information Systems Frontiers*, 21(1):87–107, 2019

- [BLPT19] Arthur Bit-Monnot, Francesco Leofante, Luca Pulina, and Armando Tacchella. SMT-based planning for robots in smart factories. In *Proc. of IEA/AIE*, pages 674–686, 2019

1.3.2 In preparation

The following publications, which represent the core on which Chapter 4 and Chapter 5 are based, are currently under submission.

- [LGÁTon] Francesco Leofante, Enrico Giunchiglia, Erika Ábrahám, and Armando Tacchella. Optimal planning modulo theories. Under submission

- [Leoon] Francesco Leofante. OMTPlan: a tool for optimal planning modulo theories. Under submission

1.3.3 A note on contributions by the author

All publications listed above where the result of fruitful collaborations with several colleagues, both in Aachen and in Genoa. However, for this thesis it is

necessary to somehow measure my contributions. In the following I will detail what I explicitly contributed to each publication, proceeding in chronological order. In general, I co-wrote all of the publications above, I will therefore only talk about the development of scientific results.

To start with, the encodings presented in [LÁN+17] and [LÁN+19] were developed by me together with Erika Ábrahám. Implementation and evaluation of the encodings was carried out by me, while the integration of these encodings in the online executive presented in [NLLÁ17] and [LÁN+19] was managed by Tim Niemueller from the Knowledge Based Systems group in Aachen.

The ideas discussed in [LÁT18, Leo18a, Leo18b] were the results of several discussions with my advisors, Erika Ábrah'am and Armando Tacchella.

The OMT-based implementation used in [BLPT19] was developed by myself, together with Igor Bongartz, a student working under my supervision. The paper was mostly written by Arthur Bit-Monnot.

Finally, most of the ideas presented in [LGÁTon] where the result of joint work with Enrico Giunchiglia, with useful suggestions given by my advisors. The implementation of the system used in [LGÁTon] and formally presented in [Leoon] was done entirely by me.

1.4 Additional publications

During the course of my Ph.D. studies I was also engaged in a parallel research stream, focusing on the problem of providing safety guarantees for learning-enabled systems – i.e., systems that employ machine learning algorithms in some of their components or that base their decisions on models derived using such algorithms. The following publications are the result of my research efforts in this direction.

- Francesco Leofante, Simone Vuotto, Erika Ábrahám, Armando Tacchella, and Nils Jansen. Combining static and runtime methods to achieve safe standing-up for humanoid robots. In *Proc. of ISoLA*, pages 496–514, 2016
- Francesco Leofante and Armando Tacchella. Learning in physical domains: mating safety requirements and costly sampling. In *Proc. of AI*IA*, pages 539–552, 2016

- Dario Guidotti, Francesco Leofante, Claudio Castellini, and Armando Tacchella. Repairing learned controllers with convex optimization: a case study. In *Proc. of CPAIOR*, pages 364–373, 2019

- Dario Guidotti, Francesco Leofante, Armando Tacchella, and Claudio Castellini. Improving reliability of myocontrol using formal verification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(4):564–571, 2019

Chapter 2

Background

The purpose of this chapter is to set the basic terminology and concepts that later chapters will explore in more detail. What is presented here is intended mainly for reference and it is not meant to be complete. Extensive references will be provided for all the results herewith presented. The chapter consists of four sections. The first section introduces basic concepts related to propositional logic and its satisfiability problem (SAT). The second section presents extensions of SAT to quantifier-free first-order logic formulas, with a focus on techniques for satisfiability and optimization in this setting. The third section instead provides background concepts on the core problem we deal with in this thesis, that is, numeric planning. In the last section, we formally define the planning as satisfiability paradigm and examine a number of encodings from planning to satisfiability checking.

2.1 The propositional satisfiability problem

A *propositional formula* is a Boolean combination of *propositions* (or *Boolean variables*) from a set $\mathcal{V}_{\mathbb{B}} = \{p_1 \dots p_n\}$. A propositional formula is in conjunctive normal form (CNF) if it is a finite conjunction of *clauses*, each of which corresponds to a finite disjunction of literals. A *literal* is either a proposition p or its complement (denoted by $\neg p$); in the first case, we say that l is a positive literal, and in the second, we say that l is a *negative literal*.

For example, the following formula

$$\varphi \quad := \quad (p_1 \vee p_2) \wedge (p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_3)$$

is a CNF defined over the propositional variables $\{p_1, p_2, p_3\}$.

Note that any propositional formula can be converted to an equi-satisfiable CNF propositional formula in by allowing the introduction of auxiliary variables (see, for example, [PG86]).

A *valuation* ν is a partial function from the set of propositions to either *true* or *false*. We say that ν satisfies a formula φ if $\nu(\varphi)$ is true, and ν falsifies φ if $\nu(\varphi)$ is false. In the former case, we call ν a *model* of φ.

For example, the formula from the previous example is satisfied by the valuation

$$\{p_1 \leftarrow false, p_2 \leftarrow true\}$$

The *propositional satisfiability problem* (SAT) requires determining if there exist a valuation of the variables of a Boolean formula, usually in CNF, such that the formula evaluates to true. A formula for which such a valuation exists is said to be *satisfiable*, otherwise it is *unsatisfiable*. Several algorithms and techniques have been devised to try and solve the propositional satisfiability problem efficiently. Since an in-depth discussion of the state of the art in SAT solving is outside the scope of this thesis, we refer the interested reader to [BHvMW09] for more details.

2.2 Satisfiability beyond propositional logic

Although applications in artificial intelligence, formal verification, and other areas have greatly benefited from advances in SAT, it is often the case that real-world applications require determining the satisfiability of formulas in more expressive logics than propositional logic. Also, these applications typically require not general first-order satisfiability, but rather satisfiability with respect to some background theory, which fixes the interpretations of certain predicate and function symbols. The research field concerned with determining the satisfiability of formulas with respect to some background theory is called *Satisfiability Modulo Theories* (SMT).

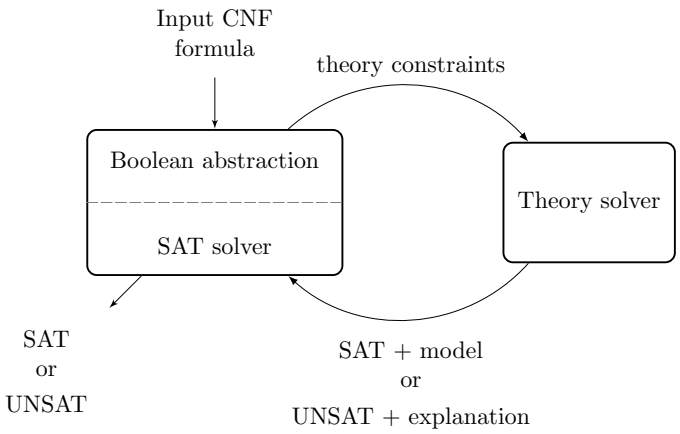


Figure 2.1: The SMT solving framework.

SMT solving aims at deciding the satisfiability of (usually quantifier-free) first-order logic formulas over some theories like, e.g., the theories of lists, arrays, bit vectors, uninterpreted functions and real or (mixed-)integer arithmetic. For the purpose of this thesis, we will deal only with quantifier-free linear integer-real arithmetic (QF_LIRA).

To decide the satisfiability of an input formula φ in CNF, SMT solvers proceed as depicted in Figure 2.1. Typically, a *Boolean abstraction* $abs(\varphi)$ of φ is built first by replacing each theory constraint with a fresh Boolean variable.

For example, the following SMT formula over $x, y \in \mathbb{R}$

$$\varphi := x \geq y \wedge (\ y > 0 \vee x > 0 \ ) \wedge y \leq 0$$

is abstracted to

$$abs(\varphi) = p_1 \wedge (\ p_2 \vee p_3 \ ) \wedge \neg p_2$$

where $p_1, p_2, p_3 \in \mathbb{B}$.

After this first step, a SAT solver is called to search for a satisfying valuation for $abs(\varphi)$, e.g., $\{p_1 \leftarrow true, p_2 \leftarrow false, p_3 \leftarrow true\}$ for the above example. If no such assignment exists then the input formula φ is unsatisfiable. Otherwise, the consistency of the assignment in the underlying theory is checked by a *theory*

solver. In our example, we check whether the set $\{x \geq y,\ y \leq 0,\ x > 0\}$ of linear inequalities is feasible, which is the case. If the constraints are theory-consistent then a satisfying solution is found for φ. Otherwise, the theory solver returns a theory lemma φ_E giving an *explanation* for the conflict, e.g., the negated conjunction of some inconsistent input constraints. The explanation is used to refine the Boolean abstraction $abs(\varphi)$ to $abs(\varphi) \wedge abs(\varphi_E)$. These steps are iteratively executed until either a theory-consistent Boolean assignment is found, or no more Boolean satisfying assignments exist.

2.3 Optimization Modulo Theories

In the last decade SMT solvers have enjoyed considerable success in many fields of application, and recently standard decision procedures for SMT have been extended to optimization, leading to the development of *Optimization Modulo Theories* (*OMT*) – see for example [NO06, CFG+10] and [BPF15, CKJ+15, ST15b, ST15c] for related solvers.

OMT extends SMT solving with optimization procedures to find a variable assignment that defines an optimal value for an objective function f (or a combination of multiple objective functions) under all models of a formula φ. This thesis is concerned with objectives expressed in QF_LIRA, however state-of-the-art solvers support optimization under other theories, such as, the theory of bit-vectors [NR16].

As described in [ST15a], most OMT solvers implement a *linear-search* scheme, which can be summarized as follows. Let φ_S be the conjunction of all theory constraints that are true under a satisfying valuation ν and the negation of those that are false under ν. A local optimum μ for f is computed under the side condition φ_ν using specialized algorithms – e.g., the simplex [Dan02] and the branch-and-bound method [LD60] when problems are expressed in QF_LIRA. The original formula φ is then updated as

$$\varphi := \varphi \wedge (f \bowtie \mu) \wedge \neg\varphi_S \quad , \quad \bowtie \in \{<, >\}$$

This forces the solver to find a new assignment under which the value of the objective function improves, while discarding all previously found assignments.

$$
\begin{array}{lcl}
e_{\mathbb{B}} & ::= & v_{\mathbb{B}} \mid \neg v_{\mathbb{B}} \\
e_{\mathbb{Q}} & ::= & \mathtt{const}_{\mathbb{Q}} \mid v_{\mathbb{Q}} \mid (e_{\mathbb{Q}} + e_{\mathbb{Q}}) \mid (e_{\mathbb{Q}} - e_{\mathbb{Q}}) \mid \\
 & & (e_{\mathbb{Q}} * e_{\mathbb{Q}}) \mid (e_{\mathbb{Q}} / e_{\mathbb{Q}}) \\
\varphi & ::= & e_{\mathbb{B}} \mid e_{\mathbb{Q}} \sim e_{\mathbb{Q}} \\
\varphi^{+} & ::= & \varphi \mid \varphi^{+}, \varphi \\
\Phi & ::= & \{\} \mid \{\varphi^{+}\} \\
\psi & ::= & v_{\mathbb{B}} := e_{\mathbb{B}} \mid v_{\mathbb{Q}} := e_{\mathbb{Q}} \\
\psi^{+} & ::= & \psi \mid \psi^{+}, \psi \\
\Psi & ::= & \{\} \mid \{\psi^{+}\}
\end{array}
$$

Figure 2.2: Abstract syntax of propositional and numeric conditions and effects. Notation: $\mathbb{B}$ is the set of the Boolean values; $\mathbb{Q}$ is the set of all rational numbers; $\mathtt{const}_{\mathbb{Q}}$ is a constant ot type $\mathbb{Q}$; v_D is a variable with domain $D \in \{\mathbb{B}, \mathbb{Q}\}$; $\sim \in \{<, \leq, =, \geq, >\}$.

Repeating this procedure until the formula becomes unsatisfiable will lead to an assignment optimizing f under all models of φ.

2.4 Planning with numeric features

As mentioned in the introduction to this document, our work is concerned with planning problems that make use of arithmetic theories. More precisely, we consider the quantifier-free fragment of numeric planning expressible in PDDL2.1 [FL03]. Although our approach could handle general quantifier-free arithmetic formulas, in the following we restrict ourselves to the expressivity of PDDL. For the sake of clarity we use a modified syntax shown in Figure 2.2 and formalize the corresponding semantics is Figure 2.3.

A *numeric planning problem* is a tuple $\Pi = \langle \mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I, G \rangle$ whose components are described in the following.

Variables and states $\mathcal{V}_{\mathbb{B}}$ and $\mathcal{V}_{\mathbb{Q}}$ are finite disjoint sets of *propositional* respectively *numeric variables* of Π. In the following variables will also be referred to as *fluents*. For a variable $v \in \mathcal{V} = \mathcal{V}_{\mathbb{B}} \cup \mathcal{V}_{\mathbb{Q}}$ let $dom(v)$ denote the *domain of* v; we use Boolean $\mathbb{B}$ for propositional variables and for numeric variables the

$$
\begin{array}{ll}
[\![v_{\mathbb{B}}]\!]_s & = s(v_{\mathbb{B}}) \\
[\![\neg v_{\mathbb{B}}]\!]_s & = \textit{if } s(v_{\mathbb{B}}) = \textit{false then true else false} \\
[\![\mathtt{const}_{\mathbb{Q}}]\!]_s & = \mathit{const}_{\mathbb{Q}} \\
[\![v_{\mathbb{Q}}]\!]_s & = s(v_{\mathbb{Q}}) \\
[\![e_{\mathbb{Q},1} + e_{\mathbb{Q},2}]\!]_s & = \textit{if } [\![e_{\mathbb{Q},1}]\!]_s \textit{ and } [\![e_{\mathbb{Q},2}]\!]_s \textit{ are defined then} \\
 & \quad [\![e_{\mathbb{Q},1}]\!]_s +_{\mathbb{Q}} [\![e_{\mathbb{Q},2}]\!]_s \textit{ else undefined} \\
[\![e_{\mathbb{Q},1} - e_{\mathbb{Q},2}]\!]_s & = \textit{if } [\![e_{\mathbb{Q},1}]\!]_s \textit{ and } [\![e_{\mathbb{Q},2}]\!]_s \textit{ are defined then} \\
 & \quad [\![e_{\mathbb{Q},1}]\!]_s -_{\mathbb{Q}} [\![e_{\mathbb{Q},2}]\!]_s \textit{ else undefined} \\
[\![e_{\mathbb{Q},1} * e_{\mathbb{Q},2}]\!]_s & = \textit{if } [\![e_{\mathbb{Q},1}]\!]_s \textit{ and } [\![e_{\mathbb{Q},2}]\!]_s \textit{ are defined then} \\
 & \quad [\![e_{\mathbb{Q},1}]\!]_s *_{\mathbb{Q}} [\![e_{\mathbb{Q},2}]\!]_s \textit{ else undefined} \\
[\![e_{\mathbb{Q},1}/e_{\mathbb{Q},2}]\!]_s & = \textit{if } [\![e_{\mathbb{Q},1}]\!]_s \textit{ and } [\![e_{\mathbb{Q},2}]\!]_s \textit{ are defined and} \\
 & \quad [\![e_{\mathbb{Q},2}]\!]_s \neq 0 \textit{ then} \\
 & \quad [\![e_{\mathbb{Q},1}]\!]_s /_{\mathbb{Q}} [\![e_{\mathbb{Q},2}]\!]_s \textit{ else undefined} \\
s \models e_{\mathbb{B}} & \textit{iff } [\![e_{\mathbb{B}}]\!]_s = \textit{true} \\
s \models e_{\mathbb{Q},1} \sim e_{\mathbb{Q},2} & \textit{iff } [\![e_{\mathbb{Q},1}]\!]_s \textit{ and } [\![e_{\mathbb{Q},2}]\!]_s \textit{ are defined and} \\
 & \quad [\![e_{\mathbb{Q},1}]\!]_s \sim_{\mathbb{Q}} [\![e_{\mathbb{Q},2}]\!]_s \\
s \models \{\varphi_1, \ldots, \varphi_n\} & \textit{iff } s \models \varphi_i \textit{ for } i = 1, \ldots, n \\
s, s' \models v_{\mathbb{B}} := e_{\mathbb{B}} & \textit{iff } s'(v_{\mathbb{B}}) = [\![e_{\mathbb{B}}]\!]_s \\
s, s' \models v_{\mathbb{Q}} := e_{\mathbb{Q}} & \textit{iff } [\![e_{\mathbb{Q}}]\!]_s \textit{ is defined and } s'(v_{\mathbb{Q}}) = [\![e_{\mathbb{Q}}]\!]_s \\
s, s' \models \{v_1 := e_1, \ldots, v_n := e_n\} & \textit{iff } s, s' \models v_i := e_i \textit{ for all} \\
 & \quad i = 1, \ldots, n \textit{ and } s(v) = s'(v) \textit{ for} \\
 & \quad \textit{all } v \in (\mathcal{V}_{\mathbb{B}} \cup \mathcal{V}_{\mathbb{Q}}) \setminus \{v_1, \ldots, v_n\}
\end{array}
$$

Figure 2.3: Semantics of propositional and numeric conditions and effects. Notation: $s, s' \in S$ are states; $\mathit{const}_{\mathbb{Q}}$ is the value of $\mathtt{const}_{\mathbb{Q}}$; $+_{\mathbb{Q}}$, $-_{\mathbb{Q}}$, $*_{\mathbb{Q}}$, $/_{\mathbb{Q}}$ and $\sim_{\mathbb{Q}}$ are addition, subtraction, multiplication, division and comparison in the domain $\mathbb{Q}$; the rest is as in Fig. 2.2.

rationals $\mathbb{Q}$ equipped with the usual order and arithmetic operations. A *state* of Π is a function $s : (\mathcal{V}_{\mathbb{B}} \cup \mathcal{V}_{\mathbb{Q}}) \to (\mathbb{B} \cup \mathbb{Q})$ assigning to each variable $v \in \mathcal{V}_{\mathbb{B}} \cup \mathcal{V}_{\mathbb{Q}}$ a value $s(v) \in \mathit{dom}(v)$ from its corresponding domain. Let S denote the set of all states of Π.

Constraints and conditions A *propositional constraint* is either a propositional variable $v \in \mathcal{V}_{\mathbb{B}}$ or its negation $\neg v$. *Arithmetic expressions* e are composed from numeric variables and constants using arithmetic operators. An arithmetic expression is *linear* iff for each multiplication operator in it, at least one of the operands contains no variables. A *numeric constraint* $e_1 \sim e_2$ compares two arithmetic expressions using a comparison operator $\sim \in \{<, \leq, =, \geq, >\}$. A *constraint* φ is either a propositional or a numeric constraint. A *condition* Φ is a (possibly empty) set of constraints (see Figure 2.2).

The evaluation function $[\![\cdot]\!]_s$ and the satisfaction relation $\models$ for expressions, constraints and conditions are as usual (see Figure 2.3).

Assignments and effects A *propositional assignment* has the form $v := e$, where $v \in \mathcal{V}_{\mathbb{B}}$ is a propositional variable and $e \in \{v, \neg v\}$. A *numeric assignment* has the form $v := e$, where $v \in \mathcal{V}_{\mathbb{Q}}$ is a numeric variable and e is an arithmetic expression (see Figure 2.2); we say that $v := e$ is an assignment *to* v. For any $v \in \mathcal{V}_{\mathbb{Q}}$, $d \in dom(v)$, and e a linear arithmetic expression, we call $v := v + d$ a *constant increment*, $v := v - d$ a *constant decrement*, $v := v + e$ a *linear increment*, and $v := v - e$ a *linear decrement*.

An *assignment* is either a propositional or a numeric assignment. An *effect* Ψ is a set of assignments that contains at most one assignment $v := e$ for each variable $v \in \mathcal{V}_{\mathbb{B}} \cup \mathcal{V}_{\mathbb{Q}}$; we say that v is *assigned* in Ψ iff there is an assignment $v := e$ in Ψ.

Given a state $s \in S$ and an effect Ψ, the *successor* of s and Ψ is the (unique) state $s' \in S$ such that $s'(v) = [\![e]\!]_s$ for each $v := e$ in Ψ, and $s'(v) = s(v)$ for each $v \in \mathcal{V}_{\mathbb{B}} \cup \mathcal{V}_{\mathbb{Q}}$ that is not assigned in Ψ; we write $s, s' \models \Psi$ (see Figure 2.3).

Actions A is a set of actions $a = (pre_a, \mathit{eff}_a, c_a)$, where pre_a is a condition, eff_a is an effect and $c_a : S \to \mathbb{Q}_{\geq 1}$ is a state-dependent positive *cost function*, specified by a numeric expression. An action $a = (pre_a, \mathit{eff}_a, c_a)$ is *applicable* in state s iff (i) $s \models \varphi$ for each $\varphi \in pre_a$ and (ii) $[\![e]\!]_s$ is defined for each assignment $v := e$ in eff_a.

A numeric constraint $e \sim 0$ is *simple* iff e is linear and for each assignment in $\cup_{a \in A} \mathit{eff}_a$, either the assigned variable does not appear in e or the assignment is a constant increment or a constant decrement.

A numeric constraint $e \sim 0$ is *linear* iff e is linear and for each assignment in $\cup_{a \in A} \mathit{eff}_a$, either the assigned variable does not appear in e or the assignment is a linear increment or a linear decrement.

A set $A' \subseteq A$ of actions is *independent* if any variable assigned in the effect of an action in the set appears in no other action in the set (neither in conditions nor in effects nor in cost functions). Being independent implies that for a fixed starting state, sequentially executing all actions leads to the same final state independently of the order in which the actions are executed. Moreover, if technically possible, independent actions could be executed concurrently, resulting in the same final state as any serial execution. We write $\oplus A' = (\cup_{a \in A'} \mathit{pre}_a, \cup_{a \in A'} \mathit{eff}_a, \sum_{a \in A'} c_a)$ to describe the condition, effect and cost of executing all actions in the independent set A'.

Initial condition I is a condition called the *initial condition* that is satisfied by exactly one state that is called the *initial state*; we assure unique satisfaction by requiring that I contains exactly one constraint for each variable, which is either v or $\neg v$ for propositional variables $v \in \mathcal{V}_{\mathbb{B}}$, and for numeric variables $v \in \mathcal{V}_{\mathbb{Q}}$ it has the form $v = d$ for some $d \in \mathit{dom}(v)$.

Goal condition and plan G is a condition called the *goal condition*; states satisfying G are called *goal states*.

A *serial plan* $\pi = \langle a_0, \ldots, a_{n-1} \rangle$ is a sequence of actions $a_0, \ldots, a_{n-1} \in A$ such that there exist (unique) states $s_0, \ldots, s_n \in S$ such that $s_0 \models I$, $s_{i-1} \models \mathit{pre}_{a_{i-1}}$ and $s_{i-1}, s_i \models \mathit{eff}_{a_{i-1}}$ for each $i = 1, \ldots, n$, and $s_n \models G$; we call n the (*serial*) *length* of the plan and $s_0, \ldots, s_n$ the plan's (*serial*) *execution*. The *cost* of π_n is $C(\pi_n) = \sum_{i=0}^{n-1} c_{a_i}(s_i)$. A plan π is *optimal* for Π iff $C(\pi) \leq C(\pi')$ for all (serial) plans π' of Π.

A *parallel plan* $\pi = \langle A_0, \ldots, A_{n-1} \rangle$ for Π is a sequence of independent action sets $A_i = \{a_{i,1}, \ldots, a_{i,k_i}\} \subseteq A$, $A_i \neq \emptyset$ for $i = 0, \ldots, n-1$, such that $\langle a_{1,1}, \ldots, a_{1,i_0}, \ldots, a_{n-1,1}, \ldots, a_{n-1,i_{n-1}} \rangle$ is a plan for Π; we call n the *parallel length* and $\sum_{i=0}^{n-1} k_i$ the *serial length* of π. Note that $\langle \oplus A_0, \ldots, \oplus A_{n-1} \rangle$ is a plan for $\langle \mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, \{\oplus A_0, \ldots, \oplus A_{n-1}\}, I, G \rangle$; we call its execution the *parallel execution* of π.

Example 1. *Let us consider a* 4×4 *grid and a robot that starts at the north-west position* $(1,1)$ *and should move to the south-east position* $(4,4)$ *by movements to south, west, north or east to a neighboring cell, without leaving the grid. The robot is initially faced towards south. Movements that require turning cost twice as much energy as without. For convenience, let* $\mathtt{S}=1$, $\mathtt{W}=2$, $\mathtt{N}=3$ *and* $\mathtt{E}=4$ *encode the four possible directions of movements. The following numeric planning problem formalizes this task:* $\Pi = \langle \mathcal{V}_{\mathbb{B}}, \mathcal{V}_{\mathbb{Q}}, A, I, G\rangle$ *with*

$$
\begin{aligned}
\mathcal{V}_{\mathbb{B}} &= \emptyset \\
\mathcal{V}_{\mathbb{Q}} &= \{x, y, f\} \\
A &= \{a_{\mathtt{S}} = (\{y<4\}, \{y:=y+1, f:=\mathtt{S}\}, c_{a_{\mathtt{S}}}), \\
&\quad\ a_{\mathtt{W}} = (\{x>1\}, \{x:=x-1, f:=\mathtt{W}\}, c_{a_{\mathtt{W}}}), \\
&\quad\ a_{\mathtt{N}} = (\{y>1\}, \{y:=y-1, f:=\mathtt{N}\}, c_{a_{\mathtt{N}}}), \\
&\quad\ a_{\mathtt{E}} = (\{x<4\}, \{x:=x+1, f:=\mathtt{E}\}, c_{a_{\mathtt{E}}})\} \\
I &= \{x=1, y=1, f=\mathtt{S}\} \\
G &= \{x=4, y=4\}
\end{aligned}
$$

where $c_{a_i}(s)=1$ *if* $s(f)=i$ *and* $c_{a_i}(s)=2$ *otherwise, for any* $i\in\{\mathtt{S},\mathtt{W},\mathtt{E},\mathtt{N}\}$ *and* $s\in S$.

There is a single optimal plan $\pi=(a_{\mathtt{S}}, a_{\mathtt{S}}, a_{\mathtt{S}}, a_{\mathtt{E}}, a_{\mathtt{E}}, a_{\mathtt{E}})$ *for* Π *with plan cost* $C(\pi)=7$.

In this example, all independent action sets have a single action, because all actions assign f.

2.5 Planning as satisfiability

In this last section we briefly review the basic ideas behind satisfiability-based approaches to planning. We will examine classical state-based encodings and briefly discuss other encodings. Although the encodings presented were originally thought for classical propositional planning, it is easy to see that the approach can be extended to SMT trivially. Hence, we will directly present the SMT version and refer the reader to [Rin09] for an in-depth study of propositional encodings. Further, non-trivial, extensions are required for OMT planning and these will be discussed in subsequent chapters.

The satisfiability approach to planning solves a planning problem Π by con-

structing sequences of logical formulas that encode *bounded versions* of it. For a given horizon n, a formula Π_n is defined whose solutions, if any, correspond to plans of length n. In most existing domain-independent approaches to planning as satisfiability, the variables over which Π_n is defined represent for each step from 0 to n the value of each fluent in Π, and for each step from 0 to $n-1$ the actions that are executed. However, other encodings exist and will be briefly discussed later in this section.

Of crucial importance in the planning as satisfiability framework is the *strategy* according to which a horizon is selected. In the satisficing planning case, a strategy must select horizons until one is found at which there exists a plan. For instance, one could use a *ramp-up* strategy in which formulas are instantiated for increasing horizon lengths, i.e., $n = 0, 1, 2, \ldots$ until a plan is found. In the optimal case instead, a strategy must find a horizon at which a plan exists, and prove that no better plan exists for any other horizon.

2.5.1 State-based encodings

The first encoding of planning problems to propositional formulas was proposed by Kautz and Selman in [KS92] in 1992. In their work, the authors presented a series of hand-crafted axioms to encode serial plans to propositional logic. This work was later extended in [KS96, KMS96] where a compilation of planning problems described in the STRIPS [FN71] formalism to propositional satisfiability is presented. These works present a number of crucial improvements to the 1992 paper, namely, parallel encodings as well as the use of *explanatory frame axioms* instead of classical frame axioms. In the following we will focus on classical encodings with explanatory frame axioms, under serial and parallel execution semantics.

The encoding of a planning problem Π for a given horizon n is given in terms of axiom schemata. The schemata make use of n variable sets $A_0, \ldots, A_{n-1}$, where each A_i consists of a unique variable a_i for each action $a \in A$, and also $n+1$ copies $\mathcal{V}_0, \ldots, \mathcal{V}_n$ of state variable sets, i.e., $\mathcal{V}_i = \{v_i | v \in \mathcal{V} = \mathcal{V}_{\mathbb{B}} \cup \mathcal{V}_{\mathbb{Q}}\}$.

Using these variables, the following formulas are defined:

- let $I(\mathcal{V}_i)$ (resp. $G(\mathcal{V}_i)$) be the formula obtained from I (resp. G) by replacing each $v \in \mathcal{V}$ with the corresponding variable $v_i \in \mathcal{V}_i$;

- let $T(\mathcal{V}_i, \mathcal{V}_{i+1})$, be a formula describing how actions affect states, i.e., T enforces that each action implies its preconditions over $\mathcal{V}_i$ and its effects over $\mathcal{V}_{i+1}$. With T we also encode explanatory frame axioms specifying that variables retain their values unless they are explicitly modified by an action's effect and *mutex* that specify the execution semantics. In serial encodings, at most one action can be executed at a time. In parallel encodings this condition is relaxed and multiple actions can be performed simultaneously provided that they are independent.

For a given planning task Π, we encode bounded plans for horizon n by the formula

$$\Pi_n := I(\mathcal{V}_0) \wedge \bigwedge_{i=0}^{n-1} T(\mathcal{V}_i, \mathcal{V}_{i+1}) \wedge G(\mathcal{V}_n)$$

By construction Π_n is satisfiable iff there exists a plan π_n within horizon n, which can be extracted from the model of the planning formula.

2.5.2 Other encodings

A number of other reductions from planning to satisfiability checking have been proposed that fall outside the scope of the previous section.

These include encodings such as those of [KMS96, KS96] and later [RGPS09] which use a split action representation to reduce the number of variables needed to describe actions and mitigate this size blow-up of direct encodings of planning problems.

Another line of work which is not covered here is that of [RHN06], which use a relaxed action execution semantics to allow greater action parallelism. The approach presented by Rintanen *et al.* leverages the concept of post-serialisability of [DNK97] allowing a set of conflicting actions to be selected for execution at the same step, provided there exist a scheme, computed a priori, that allows the generation of a valid serial execution. In practice, this is achieved by replacing the standard conflict exclusion axioms with axioms that ensure that conflicting parallel actions respect the serialization scheme.

Finally, we draw attention to the work that has been done in causal encoding. These are based on the idea of proving the correctness of a plan using causal

reasoning about the establishment and preservation of goals and of the preconditions of individual actions. Causal encodings were first presented in [KMS96] while a theoretical and empirical study of their properties is discussed in [MK99].

Chapter 3

Planning for logistics

With the advent of Industry 4.0, factories are moving from static process chains towards the introduction of autonomous robots in their production lines. As the abilities and the complexity of such systems increase, the problem of managing and optimizing the in-factory supply chain carried out by (fleets of) autonomous robots becomes crucial. This paradigm shift also opens up a number of new research challenges for the planning community.

The *RoboCup Logistics League* (RCLL) [NLF15] has been proposed as a realistic testbed to study the above mentioned problems at a manageable scale. There, groups of robots need to maintain and optimize the material flow according to dynamic orders in a simplified factory environment.

Though there exist successful heuristic methods to solve the planning and scheduling problem underlying the RCLL, e.g., [HNCL16, NLF13], a major disadvantage of these methods is that they provide *no guarantees* about the quality of the solutions they produce. In this chapter we present our efforts towards solving this problem leveraging OMT. In particular, we propose domain-specific OMT encodings for task planning with optimality guarantees and integrate them into an on-line execution and monitoring system based on CLIPS [Wyg89], a rule-based production system using forward chaining inference.

Before delving into the details of our encodings, we describe the salient features of the RoboCup Logistics League. We then provide a high-level description of the architecture of a system we developed for integrated planning and exe-

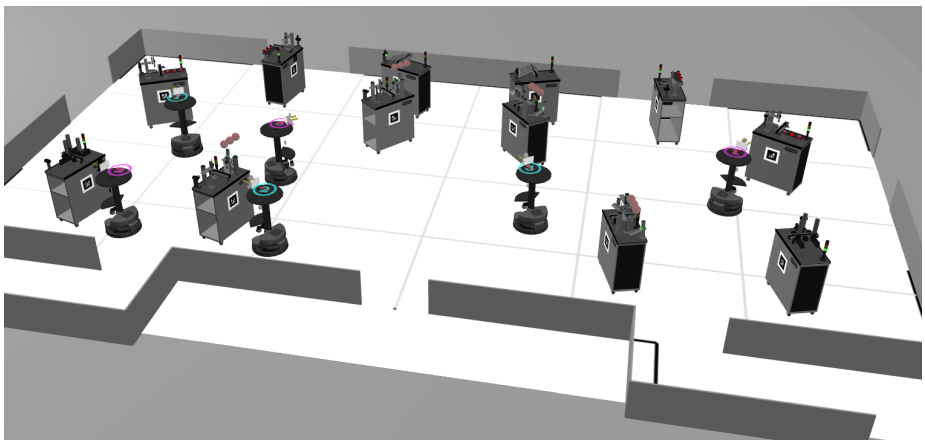


Figure 3.1: Simulated RCLL factory environment [ZNL14].

cution. Our OMT encodings for the different phases of the RCLL, together with their experimental validation, are presented; we conclude with preliminary experiments concerning plan explainability in the RCLL setup.

3.1 The RoboCup Logistics League

The example domain chosen for evaluating our planning and execution approach is based on the *Planning and Execution Competition for Logistics Robots in Simulation*[1] [NKVT16], which provides a simulated version of the real RCLL setup (Figure 3.1). During a game in the competition, autonomous robots compete to handle the logistics of materials through several dynamic stages to produce final goods according to a dynamic order schedule known only at runtime. Each game sees two teams of three robots each competing against each other during two phases, the *exploration* and the *production* phase.

In the exploration phase, robots must roam the environment and determine where the team's own machines are positioned. For this, the playing field is divided into 24 virtual zones, of which 12 belong to each of the two teams (operating at the same time in the environment increasing execution duration uncertainty considerably). However, only 6 of these zones will contain machines. Therefore, the task is to efficiently assign the three robots to the 12 zones and

[1] `http://www.robocup-logistics.org/sim-comp`

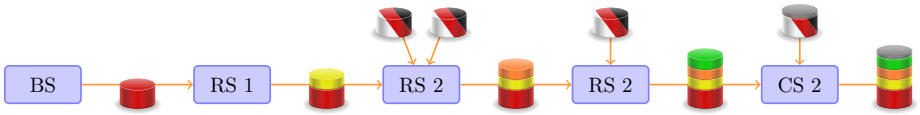


Figure 3.2: Example of order configuration for the competition [NLF15, RCL17].

identify the zones which contain a machine.

In the production phase instead, robots have to handle the logistics of materials through several (dynamic) stages to produce final goods to fulfill orders. Products to be assembled have different complexities and usually require a base, 0 to 4 rings to be mounted on top of it, and a cap as a finishing touch. To increase complexity, orders not only fix the components to be used, but also specify colors to be used, and in what order. Bases are available in three different colors, four colors are admissible for rings and two for caps.

Several machines are scattered around the factory shop floor, each of them completing a different processing step such as providing bases, mounting colored rings or caps. Based on such differences, it is possible to distinguish four types of machines:

- Base Station (BS): acts as dispenser of base elements. There is one single BS per team.
- Cap Station (CS): mounts a cap as the final step in production on an intermediate product. CS have a slide to store at most one cap piece at a time. At the beginning of the game this slide is empty and has to be filled as follows. A base element with a cap must be taken from a shelf in the game arena and fed to the machine; the cap is then unmounted and buffered in the slide. The cap can then be mounted on the next intermediate product taken to the machine. There are two CS per team.
- Ring Station (RS): mounts one colored ring (of a specific color) onto an intermediate product. Some ring colors require additional tokens to be "unlocked": robots will have to feed a RS with a specified number of bases before the required color can be mounted. There are two RS per team.

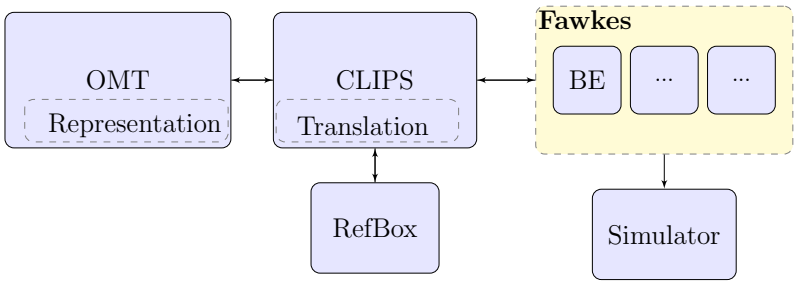


Figure 3.3: High-level representation of how individual components of the system interact.

- Delivery Station (DS): accepts finished products. A DS contains three conveyor belts, robots have to prepare the proper one as per specific order. There is one DS per team.

The challenge for autonomous robots is then to transport intermediate products between processing machines and optimize a multistage production cycle of different product variants until delivery of final products. A sample production trace is shown in Figure 3.2.

Orders that denote the products which must be assembled with these operations are posted at run-time by an automated referee box (RefBox) broadcasting information via Wi-Fi and therefore require quick planning and scheduling. Orders come with a delivery time window introducing a temporal component into the problem.

3.2 System overview

The system described in this section unites the power of Optimization Modulo Theories with the flexibility of an on-line executive, providing optimal solutions for high-level task planning, and runtime feedback on their feasibility. The proposed architecture is depicted in Figure 3.3. The CLIPS agent controls the overall process, from the generation of a plan, to its execution and monitoring. When a new plan is needed, the agent triggers the OMT module to synthesize a plan. To start planning, the world model, with all relevant information, must be encoded in a way accessible to the OMT solver. We have opted for Google Protocol Buffers (`protobuf`) to handle communications to and from the OMT

solving module. Once a plan is computed, CLIPS retrieves it and distributes it to the robots for execution. Robots then execute their respective partial plans by invoking the appropriate basic behaviors through the behavioral and functional components of the Fawkes[2] software framework (for instance, BE in Figure 3.3 represents the Lua-based Behavior Engine [NFL09] that provides the basic skills to execute plans). Only through this framework does the reasoning system interact with the simulation.

Several challenges can arise during execution, as original modeling assumptions might not hold in the real system due to, e.g., action failure, plan failure due to ignorance or change in a dynamic environment. If this happens, plans might become inconsistent and lead to undesired behaviors. In our framework, we rely on the interplay between the planning module and the on-line executive to tackle this problem. Once plans have been generated, CLIPS automatically starts the appropriate tasks. Updates on execution (e.g., if a certain task is currently in progress, task failures) are always distributed in the world model, therefore the executive is constantly informed about execution progress. When inconsistencies with the model are detected, the executive can ask for a new plan, and our encoding allows to compute this starting from any arbitrary initial world state.

In the following, we describe the main components of our system and show how they operate together in our pipeline.

3.2.1 CLIPS rules engine

The "C" Language Production System (CLIPS) [Wyg89] is a rule-based production system developed at NASA which uses forward chaining inference based on the Rete algorithm [For82]. CLIPS consists of three building blocks [Gia07]: a *fact base*, a *knowledge base* and an *inference engine*.

The fact base can be seen as a global memory where data is stored in the form of *facts*, high-level statements that encode pieces of information about the world state. The knowledge base instead, is used to represent knowledge. More specifically, CLIPS provides heuristic and procedural paradigms for representing

[2]Fawkes is a component-based software framework for robotic real-time applications. URL: www.fawkesrobotics.org

knowledge in the form of *rules* and *functions* respectively.

Rules specify heuristics to decide which actions to perform in what situations. An example of a CLIPS rule is shown in Listing 3.1. Formally, rules are composed of an *antecedent* and a *consequent*. The antecedent is defined as a set of conditions expressed over facts (lines 2–6), while the consequent consists of a set of *actions* to be performed (lines 8–16) when the rule is applicable. Actions in CLIPS are represented by functions (lines 8–14, `omt-create-*` calls are functions), pieces of executable code which can return values or perform side-effects (e.g., interact with the low-level control layer for robots).

The inference engine is the mechanism that CLIPS provides to control the overall execution of rules. At system initialization, the inference engine is instructed to begin execution of applicable rules. To determine whether a rule is applicable, the inference engine checks for each rule in the knowledge base whether their antecedent is met by the facts initially asserted in the fact base.

If all conditions specified in the antecedent of a rule are satisfied then the rule is activated and added to the execution agenda. If more than one rule is applicable, the inference engine uses a *conflict resolution* strategy to select which rule should have its actions executed. The actions of the selected rule are executed (which may affect the list of applicable rules) and then the inference engine selects another rule and executes its actions. This process continues until no applicable rules remain.

3.2.2 Communication infrastructure

For plan synthesis, the world model, with all relevant information, must be encoded in a way accessible to the solver. In this work, we have used Google Protocol Buffers[3] (`protobuf`) to encode the world state when synthesis is triggered, as well as the resulting plan. Protocol buffers define a language-independent mechanism for serializing structured data. To use them, one needs to specify the structure of the data to be serialized (i.e., specify the data type). Once this is done, the protocol buffer compiler needs to be run to automatically generate data access classes in the language of interests – C++ in our case. Protobuf buffers provide a convenient transport, exchange, and storage representation

[3] `https://developers.google.com/protocol-buffers/`

```
1 (defrule production-call-clips-omt
2 (phase EXPLORATION)
3 (team-color ?team-color&CYAN|MAGENTA)
4 (state IDLE)
5 (not (plan-requested))
6 (test (eq ?*ROBOT-NAME* "R-1"))
7 =>
8 (bind ?p
9 (omt-create-data
10 (omt-create-robots ?team-color)
11 (omt-create-machines ?team-color)
12 (omt-create-orders ?team-color)
13 )
14 )
15 (omt-request "explore-zones" ?p)
16 (assert (plan-requested))
17 )
```

Listing 3.1: CLIPS rule to trigger synthesis.

that is easy to create and read. They also have powerful introspection capabilities which are particularly useful for generic access from reasoning systems. For example, the CLIPS-based access requires only the message definition files and not any pre-generated code. We use the exploration problem as a working example to show the interaction between the solving module and the CLIPS agent. The rule to trigger the synthesis process is shown in Listing 3.1. Once the game is started (lines 2–4), the first robot (line 6) will create a data structure initialized with all relevant information needed to compute a plan (lines 8-14), and pass it over to the OMT solver to request a plan (line 15).

The OMT side uses this data to build planning encodings (more details on these are given in the following sections). If solving completes successfully, the OMT plug-in notifies the executive that a solution is ready for retrieval. An excerpt of the message specifications for plan representation is shown in Listing 3.2. First, a list of actor (robot) specific plans is defined in lines 1–3, where the keyword `repeated` specifies that the field may be repeated multiple times. Each plan (lines 4–11) requires the actor for the plan to be defined (`required` keyword) and either is a serial or a temporal plan (`oneof` keyword). In this example, we show how a message for serial plans is defined (lines 12–14).

```
1 message ActorGroupPlan {
2 repeated ActorSpecificPlan plans = 1;
3 }
4 message ActorSpecificPlan {
5 required string  actor_name = 1;
6
7 oneof plan {
8 SequentialPlan sequential_plan = 2;
9 TemporalPlan   temporal_plan = 3;
10 }
11 }
12 message SequentialPlan {
13 repeated PlanAction actions = 1;
14 }
15 message PlanAction {
16 required string name = 1;
17 repeated PlanActionParameter params = 2;
18 }
19 message PlanActionParameter {
20 required string key = 1;
21 required string value = 2;
22 }
```

Listing 3.2: Plan data type specfication in protobuf. Each field requires a numerical tag, that identifies the field in the binary encoding.

A serial plan simply consists of a series of actions (lines 15–18), each of which is defined by a name and parameterized by a number of key-value pairs (lines 19–22). Listing 3.3 shows a concrete example of a plan for two robots – `"R-1"` and `"R-2"` – with two `"move"` action commands.

3.2.3 Execution and monitoring

Once a plan has been retrieved, it must be translated into a native CLIPS representation. Each action specified by the OMT module (see Listing 3.3) is added to the fact base by means of facts which identify tasks and steps to be executed on the CLIPS side. Rules are defined to process such tasks and steps, defining the actions to be executed. Listing 3.4 shows an example of such a translation for Listing 3.3, lines 1–19. First, a task fact is added (lines 1–2) to

```
plans[0]:ActorSpecificPlan {
actor_name: "R-1"
sequential_plan:SequentialPlan {
actions[0]:PlanAction {
name: "move"
params[0]:PlanActionParameter {
key: "to"
value: "C-BS-I"
}
}
actions[1]:PlanAction {
name: "move"
params[0]:PlanActionParameter {
key: "to"
value: "C-DS-I"
}
}
}
}
plans[1]:ActorSpecificPlan {
actor_name: "R-2"
sequential_plan:SequentialPlan {
actions[0]:PlanAction {
name: "move"
params[0]:PlanActionParameter {
key: "to"
value: "C-CS1-I"
}
}
actions[1]:PlanAction {
name: "move"
params[0]:PlanActionParameter {
key: "to"
value: "C-RS2-I"
}
}
}
}
```

Listing 3.3: Plan represented through the messages from Listing 3.2 (shown in augmented JavaScript Object Notation).

```
1 (task (task-id 1910) (robot "R-1") (name explore)
2 (state proposed) (steps 1911 1912))
3 (step (id 1911) (name drive-to) (state inactive)
4 (machine C-BS) (side INPUT)
5 (sync-id (next-sync-id)))
6 (step (id 1912) (name drive-to) (state inactive)
7 (machine C-DS) (side INPUT)
8 (sync-id (next-sync-id)))
```

Listing 3.4: Task representation in CLIPS.

specify robot actor and steps to be executed. Step facts are specified in lines 2–8, where more details about the low-level robot actions are added.

After a plan is added to the fact base, it must be distributed to all robots for execution. To do so, our system relies on the communication infrastructure used to share world model updates among the robots. This encapsulates fact base updates in protobuf messages and broadcasts them to the other robots. A (dynamically elected) master generates a consistent view and distributes it to the robots. On each robot, the CLIPS executive has rules that automatically start tasks when applicable. Basic behaviors in our framework are provided by a Lua-based Behavior Engine, but could in principle be provided by other sources. A step in a task is executed by triggering the execution of an asynchronous durative procedure. Then, information about the execution of the state is read and asserted in the fact base. Updates on task execution (e.g., whether a task is currently in progress) are distributed in the world model, making sure that the on-line executive is informed about the status of execution.

During execution, the modeling assumptions may be challenged and, in general, actions may fail or produce an unexpected result. For instance, an object might be misplaced, or slack during execution could make a plan invalid, for example if a specified deadline cannot be met. As explained above, steps of a task are triggered non-blocking, i.e., rule evaluation continues normally. This can be used to implement execution monitoring, where rules can be defined to identify situations where a step should be skipped or a task be aborted.

3.3 Exploration phase

In this section we show how to construct plans for the *exploration phase* of a game in the RCLL. Although exploration does not play a major role in determining the outcome of a competition, we decided to start with this phase because of the easy formulation of the underlying problem. As explained in Section 3.1, in the exploration phase the robots must roam the environment and determine where the team's own machines are positioned. Each team is assigned 12 virtual zones to explore, out of which only 6 contain machines. However, even though the problem formulation looks simple, computing an optimal solution (in terms of fastest execution) proved to be challenging: optimal exploration is a variant of the multiple traveling salesman problem, which is known to be NP-hard. As we learned, the combinatorial nature of this problem poses a great challenge to the OMT solver: naive encodings fail to cope with the complexity of the domain.

The experimental analysis presented here has been carried out using the νZ solver[4] [BPF15]. Though most of the encodings we present in the following generate linear arithmetic problems, due to the Boolean structure of these formulas we could not use any linear programming tools. We considered also the OMT solvers SMT-RAT [CKJ+15] and OptiMathsat [ST15b]. The latter specializes in optimization for real arithmetic problems, whereas SMT-RAT is tuned for the satisfiability check of non-linear real arithmetic formulas. However, the nature of our problems rather requires combinatorial optimization at the Boolean level and therefore the strengths of these two solvers could not be exploited to their fullest. νZ was the tool which could solve all the instances proposed, therefore it was chosen as best candidate for our empirical analysis.

3.3.1 Developing and testing different encodings

We will now proceed with a presentation of the different encodings we proposed to solve the exploration phase. These encodings, being domain-specific, deviate substantially from the standard state-based representations of Chapter 2 and are strongly optimized toward the specific problem considered.

[4]Running on a machine running Ubuntu Mate 16.4, Intel Core i7 CPU at 2.10GHz and 8GB of RAM

$$\varphi_{depot} := \left\{ \begin{array}{l} pos_{1,-1} = -1 \wedge pos_{1,0} = 0 \wedge pos_{2,-2} = -2 \wedge \\ pos_{2,-1} = -1 \wedge pos_{2,0} = 0 \ \wedge pos_{3,-3} = -3 \wedge \\ pos_{3,-2} = -2 \wedge pos_{3,-1} = -1 \wedge pos_{3,0} = 0 \end{array} \right. \tag{3.1}$$

$$\varphi_{move} := \left\{ \begin{array}{l} \bigwedge_{i=1}^{3} d_{i,0} = 0 \ \wedge \\ \bigwedge_{j=1}^{Z} \left[\left(\bigvee_{k=0}^{Z} \bigvee_{\substack{l=1 \\ l \neq k}}^{Z} pos_{i,j-1}{=}k \wedge pos_{i,j}{=}l \wedge d_{i,j}{=}d_{i,j-1}{+}D(k,l) \right) \right. \\ \left. \vee \left(pos_{i,j}{=}-4 \wedge d_{i,Z}{=}d_{i,j-1} \right) \right] \end{array} \right. \tag{3.2}$$

$$\varphi_{each} := \left\{ \bigwedge_{k=1}^{Z} \left[\bigvee_{i=1}^{3} \bigvee_{j=1}^{Z} \left(pos_{i,j} = k \wedge \bigwedge_{u=1}^{3} \bigwedge_{\substack{v=1 \\ (u,v) \neq (i,j)}}^{Z} pos_{u,v} \neq k \right) \right] \right. \tag{3.3}$$

$$\varphi_{max} := \left\{ \bigwedge_{i=1}^{3} \left[m_i \Leftrightarrow \left(\bigwedge_{\substack{l=1 \\ l<i}}^{3} d_{l,Z} < d_{i,Z} \wedge \bigwedge_{\substack{l=1 \\ i<l}}^{3} d_{l,Z} = d_{i,Z} \right) \right] \right. \tag{3.4}$$

Figure 3.4: SMT encoding (A) for the exploration phase.

First encoding (A). We encode the high-level task to explore Z zones by 3 robots as shown in Figure 3.4. Robots start from a depot, modeled by some *fictitious zones* $-3, -2, -1$. Each robot $i \in \{1, 2, 3\}$ starts at zone $-i$, moves over to the zones $-i{+}1, \ldots, 0$, and explores, from the *start zone* 0, at most Z of the zones $1, \ldots, Z$. The distance between two zones i and j is denoted by $D(i, j)$. Here we assume the distance that a robot needs to travel to reach the start zone to be 0, but it could be also set to any positive value (see Figure 3.5).

The movements of robot i are encoded by a sequence $pos_{i,-i}, \ldots, pos_{i,Z}$ of zones it should visit, with $pos_{i,j} \in \mathbb{Z}$. The variables $pos_{i,-i}, \ldots, pos_{i,0}$ in φ_{depot} in formula (3.1) represent the movements from the depot to the start zone.

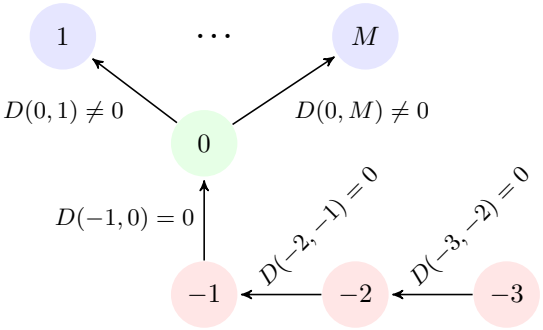


Figure 3.5: Initial robot configuration.

For $j > 0$, if the value of $pos_{i,j}$ is between 1 and Z then it encodes the jth zone visited by robot i. Otherwise, $pos_{i,j} = -4$ encodes that the robot stopped moving and stays at its last position for the rest of the exploration (i.e., the plan does not require robot i to explore any more zones). The total distance traveled by robot i to visit zones until step j is stored in $d_{i,j} \in \mathbb{R}$. These facts are encoded by φ_{move} in formula (3.2): for each robot $i \in \{1, 2, 3\}$ we set $d_{i,0} = 0$ and for each $j \in \{1, \ldots, Z\}$, we make sure that, at each step j, either the robot moves and its travel distance is incremented accordingly, or the robot stops moving. Notice that in this second case, we can immediately determine the total travel distance for the robot at the last step in the plan and, furthermore, the above constraints imply that once robot i stops moving ($pos_{i,j} = -4$) it will not move in the future ($pos_{i,j'} = -4$ and $d_{i,j'} = d_{i,j'-1}$ for all $j \leq j' \leq Z$).

For each zone $k \in \{1, \ldots, Z\}$ we enforce that it is visited exactly once by requiring φ_{each} in formula (3.3).

Finally φ_{max} in formula (3.4) uses for each robot $i \in \{1, 2, 3\}$ a Boolean variable m_i to encode whether the robot has the smallest index under all robots with maximal total travel distances at the end of their plans (note that there is exactly one robot with this property).

Our optimization objective is to minimize the largest total travel distance:

$$\text{minimize} \quad \textstyle\sum_{i=1}^{3} m_i \cdot d_{i,Z} \tag{3.5}$$

$$\textit{subject to } \varphi_{depot} \wedge \varphi_{move} \wedge \varphi_{each} \wedge \varphi_{max}$$

Results. Encoding A allowed us to compute optimal plans, but it did not scale well with the number of zones to be visited. The solving time 286.7 seconds listed in Table 3.1 for the optimal objective 12.6 for a benchmark with $Z = 12$ zones claims a large part of the overall duration of the exploration phase.

Tackling loosely connected constraints (B). By analyzing solver statistics we noticed that the number of theory conflicts was quite large, and theory conflicts typically appeared at relatively high decision levels, i.e., at late stages of the Boolean search in the SAT solver. One reason for this is that during optimization, violations of upper bounds on the total travel distances can be recognized by the theory solver only if all the zones that a robot should visit are already decided. In other words, the constraints defining the total travel distance of a robot build a loosely connected chain in their variable-dependency graph. Furthermore, explanations of the theory conflicts blamed the whole plan of a robot, instead of restricting it to prefixes that already lead to violation. As a result, the propositional search tree could not be efficiently pruned. To alleviate this problem, we added to the encoding (A) the following formula, which is implied by the monotone increment of the partial travel distances by further zone visits:

$$\bigwedge_{i=1}^{3} \bigwedge_{j=1}^{Z} d_{i,j} \leq d_{i,Z} \tag{3.6}$$

Results. As Table 3.1 shows, adding the above constraints led to a slight improvement, but the solving time of 255.55 seconds for 12 zones is not acceptable for the application considered.

Symmetry breaking (C). Although the robots start from different zones, all move to the start location 0 at cost 0 before exploration. Thus, given a schedule for the three robots, a renaming of the robots gives another schedule with the same maximal travel distance. These symmetries result in the solver covering unnecessarily redundant search space, significantly increasing solving time. However, breaking these symmetries by modifying the encoding and without

modifying the solver-internal algorithms is hard. A tiny part of these symmetries, however, can be broken by imposing on top of encoding (B) that a single, heuristically determined zone k (e.g., the closest or furthest to zone 0) should be visited by a fixed robot i:

$$\bigvee_{j=1}^{Z} pos_{i,j} = k \tag{3.7}$$

Results. This at first sight rather weak symmetry-breaking formula proved to be beneficial, resulting in a greatly reduced number of conflicts as well as solving time (81.64 seconds for $Z = 12$ zones, see Table 3.1). However, this encoding just fixes the robot that should visit a given single zone, thus the computational effort for Z zones reduces only to a value comparable to the previous effort (using encoding (B)) for $Z - 1$.

Explicit scheduler choice (D). In order to make the domain over which the variables $pos_{i,j}$ range more explicit, we added to encoding (C) the following requirement:

$$\bigwedge_{i=1}^{3} \bigwedge_{j=1}^{Z} (pos_{i,j} = -4 \vee \bigvee_{k=1}^{Z} pos_{i,j} = k) \tag{3.8}$$

Results. This addition led to some performance gain. With a solving time of 54.17 seconds for 12 zones, our approach could be successfully integrated in the RCLL planning framework.

Partial bit-blasting (E). To reduce the number and size of theory checks, we also experimented with partial bit-blasting: the theory constraints $pos_{i,j}{=}k$ in encoding (C) were replaced by Boolean propositions $pos_{i,j,k}{\in}\mathbb{B}$, which are true iff robot i visits zone k at step j. For each $i{\in}\{1,2,3\}$ and $j{\in}\{-3,\ldots,Z\}$ we ensure that there is exactly one $k{\in}\{-4,\ldots,Z\}$ for which $pos_{i,j,k}$ is true by bit-blasting for the $Z{+}5$ possible values (using fresh propositions $p_{i,j,k} \in \mathbb{B}$):

$$
\begin{aligned}
pos_{i,j,0} \quad &\Longleftrightarrow \quad (\neg p_{i,j,\lceil \log(Z+5) \rceil} \wedge \ldots \wedge \neg p_{i,j,0}) \\
pos_{i,j,1} \quad &\Longleftrightarrow \quad (\neg p_{i,j,\lceil \log(Z+5) \rceil} \wedge \ldots \wedge \; p_{i,j,0}) \quad \ldots
\end{aligned} \tag{3.9}
$$

Results. As shown in Table 3.1, partial bit-blasting did not introduce any improvement. On the contrary, an optimal solution for 12 zones could not be computed within 5 minutes. We made several other attempts to improve the running times by modifying encoding (D), but they did not bring any major improvement.

Explicit decisions (F). Even though encoding (D) could be integrated in the RCLL framework, we investigated ways to further reduce the solving times.

To this purpose, we developed a new encoding shown in Figure 3.6, in which we made some decisions explicit by means of additional variables.

In particular, for each $k \in \{1, \ldots, Z\}$ we introduced an integer variable m_k to encode which robot visits zone k, and an integer variable $n_{i,k}$ for each $i \in \{1, 2, 3\}$ to count how many of the zones $1, \ldots, k$ robot i has to visit. The meaning of these variables are encoded in φ_{visits} in formula (3.10).

We keep the position variables $pos_{i,j}$ to store which zone is visited in step j of robot i, but their domain is slightly modified: knowing the number $n_{i,Z}$ of visits for each robot, the fictitious location $pos_{i,j} = -4$ is not needed anymore. Instead, we will simply disregard all $pos_{i,j}$ assigned for $j > n_{i,Z}$.

We also keep the variables $d_{i,j}$, but with a different meaning: $d_{i,j}$ stores the distance traveled by robot i from its $(j-1)$th position $pos_{i,j-1}$ to its jth position $pos_{i,j}$. We add the constraints formula (3.11) for defining the positions up to the start zone and additionally the constraints in formula (3.12). Note that we replaced $d_{i,j} = D(k, l)$ with a weak inequality constraint. As we discuss later in this section, this was possible as the minimization of travel distances will anyways enforce the equality, but solving inequalities seems to be easier for νZ.

$$\varphi_{visits} := \left\{ \bigwedge_{i=1}^{3} \left[n_{i,0} = 0 \wedge \bigwedge_{k=1}^{Z} (m_k = i \wedge n_{i,k} = n_{i,k-1} + 1) \vee (m_k \neq i \wedge n_{i,k} = n_{i,k-1}) \right] \right. \tag{3.10}$$

$$\varphi_{depot} := \left\{ \begin{array}{l} pos_{1,-1} = -1 \wedge pos_{1,0} = 0 \wedge pos_{2,-2} = -2 \wedge \\ pos_{2,-1} = -1 \wedge pos_{2,0} = 0 \ \wedge pos_{3,-3} = -3 \wedge \\ pos_{3,-2} = -2 \wedge pos_{3,-1} = -1 \wedge pos_{3,0} = 0 \end{array} \right. \tag{3.11}$$

$$\varphi_{dist} := \left\{ \bigwedge_{i=1}^{3} \bigwedge_{j=1}^{Z} \left[\bigvee_{k=0}^{Z} \bigvee_{\substack{l=1 \\ l \neq k}}^{Z} \Big(pos_{i,j-1}{=}k \wedge pos_{i,j}{=}l \wedge d_{i,j} \geq D(k,l) \Big) \right] \right. \tag{3.12}$$

$$\varphi_{tot} := \left\{ \bigwedge_{i=1}^{3} \left[(n_{i,Z} = 0 \wedge d_i \geq 0) \vee \bigvee_{k=1}^{Z} (n_{i,Z} = k \wedge d_i \geq \sum_{j=1}^{k} d_{i,l}) \right] \right. \tag{3.13}$$

$$\varphi_{all} := \left\{ \bigwedge_{i=1}^{3} \bigwedge_{k=1}^{Z} \left[m_k = i \implies \bigvee_{j=1}^{Z} \big(n_{i,Z} \geq j \wedge pos_{i,j} = k \big) \right] \right. \tag{3.14}$$

$$\varphi_{bounds} := \left\{ \bigwedge_{i=1}^{3} \bigwedge_{k=1}^{Z} \bigwedge_{j=1}^{Z} 1 \leq m_k \leq 3 \wedge 0 \leq n_{i,k} \leq Z \wedge 1 \leq pos_{i,j} \leq Z \right. \tag{3.15}$$

$$\varphi_{symm} := \left\{ \ d_1 \geq d_2 \wedge d_2 \geq d_3 \right. \tag{3.16}$$

Figure 3.6: SMT encoding (F) for the exploration phase.

Z	(A)		(B)		(C)		(D)		(E)		(F)	
	Time	Conf	Time	Conf	Time	Conf	Time	Conf	Time	Conf	Time	Conf
6	0.40	4841	0.25	3206	0.18	2525	0.17	2069	0.29	3416	0.16	1103
8	2.07	14400	1.91	15248	1.16	9237	1.62	14355	5.32	30302	1.23	3876
10	80.06	225518	59.71	184685	26.71	91648	21.72	89785	TO		8.97	27811
12	286.70	486988	255.55	449485	81.64	198249	54.17	161134	TO		36.21	101308

Table 3.1: Running times (sec) and conflicts for encodings (A)-(F) evaluated over 100 benchmarks (Z: number of zones to be visited, TO: 5min).

A new variable d_i for $i \in \{1, 2, 3\}$ is used to store the total travel distance for each robot in φ_{tot} in formula (3.13), which ensures that, if robot i has to visit k zones ($n_{i,Z}$=k) then its total travel distance d_i is (at least equal to) the sum of the distances traveled from $pos_{i,0}$ to $pos_{i,k}$. If robot i does not move at all (i.e., $n_{i,Z} = 0$) then d_i will be (at least) zero.

The formula φ_{all} in formula (3.14) makes sure that each robot visits all zones it has been assigned to by means of variables m_k: if robot i is assigned to zone k then this zone will be visited at some step j (within the upper bound on the number of zones to be visited $n_{i,Z}$).

Furthermore, in φ_{bounds} in formula (3.15) we introduce bounds on integer variables so that the solver can represent integers as bit-vectors and internally perform bit-blasting.

Finally, we replace the nonlinear objective function specified in formula (3.5) by a linear one: since all robots start from the start zone, we exploit symmetry and require an order on the total travel distances in formula (3.16). We can now minimize the total distance for the first robot d_1 under the side condition that the conjunction of all formulas in Figure 3.6 holds.

Results. Table 3.1 shows a considerable improvement by encoding (F) over previous solutions for the selected benchmarks.

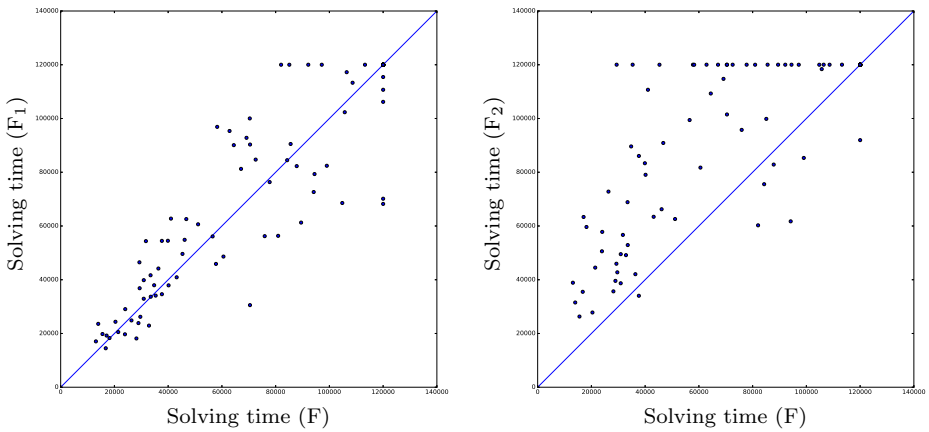
In order to obtain statistically significant results, we also tested encoding (F) on 100 most recurring instances of the RCLL problem with 12 zones (see Table 3.2). Especially the replacement of a non-linear objective function with a linear one allowed us to reduce the complexity of the optimization problem at hand.

Z	(F)		(F_1)		(F_2)	
	Time	#solved	Time	#solved	Time	#solved
12	54.78	66/100	57.02	66/100	66.84	46/100

Table 3.2: Average solving time (sec) and #instances solved for encodings (F), (F_1) and (F_2) on 100 benchmarks (TO: 2min).

To analyze the potential sources of improvement, we made additional experiments with two variants of encoding (F): in encoding (F_1) we removed the bounds for integer variables as specified in formula (3.15), and in encoding (F_2) we replaced the inequalities in formula (3.12) and (3.13) with equalities (while the constraints from formula (3.15) are kept in (F_2)). Table 3.2 and Figure 3.7 show results for the previously used 100 benchmarks.

While working with unbounded integers in encoding (F_1) does not seem to significantly affect the solving times, the solving time for the encoding (F_2) with equalities is almost always higher, and less instances could be solved within the timeout.



(a) (b)

Figure 3.7: Comparison of solving times (msec): encodings (F) versus (F_1) (left) and (F) versus (F_2) (right) ($Z = 12$, TO: 2min).

3.4 Production phase

Building on the results obtained for the exploration phase, we moved on to consider the *production phase* of the RCLL. This part of the game poses challenges to the OMT solver that are different in nature with respect to the ones met before. One the one hand, production tasks are more constrained and therefore present less symmetries than exploration. On the other hand, they require more sophisticate robot-robot and robot-environment interactions, which affect both plan synthesis and execution.

The methodology presented here has been fully integrated in the system presented in Section 3.2 and won the first place in the Planning and Execution Competition for Logistics Robots in Simulation held at the International Conference on Automated Planning and Scheduling 2018.[5]

3.4.1 Building a formal model for production processes

Given an RCLL configuration, our goal is to find a bounded sequence of robot actions that maximizes the total reward achieved for delivering ordered products. Due to the complexity of the RCLL domain, several challenges arise when building a logical encoding of this optimization problem. The formal model needs to account for concurrent robot motions, production processes and machine states, order schedules, deadlines and rewards.

As for the previous section, our encodings are non-standard. Here, however, we do follow the structure of state-based encodings to some extent. The novelty of our encoding lies in the state representation adopted. Instead of creating full copies of state variables for each planning step, we use an abstract representation that encodes salient information about the world. For instance, instead of creating variables to represent the state of all three robots, we create one single set of variables that describe the state of a generic robot. The specification of the concrete state of a given robot is then achieved using the domains of such state variables.

Such representation also requires a different encoding of the transition function: classical encodings would yield inconsistent information here. As we will

[5] http://icaps18.icaps-conference.org/

show, we alleviate this problem by encoding *long-distance* dependencies between variables. Each time a concrete resource – be it a machine or a robot – needs to be used, our encoding goes back to the last planning step where the corresponding variables were used and propagates their values to the current decision step. We now proceed with a formal characterization of how this whole process is encoded in SMT formulas.

Our reduced-size representation assumes that decisions on actions are made sequentially for one robot at a time; the transitions in T will model those decisions and their effects by updating the states of all components of the model accordingly. Continuous variables are used to keep track of time – e.g., when a robot starts an action or a machine completes a production step – and are used to ensure that decisions made locally during each step are time-consistent at a global level.

Let M represent the total number of machines in the arena, R the number of robots used and n the planning horizon considered. The first step towards defining a formal model for robot motions and machine processes is to identify a set of variables that encode all the relevant properties of the system's state. To be able to refer to the jth action and its effects, we attach an index from the domain $\{1, \ldots, n\}$ to the variables. Furthermore, since actions have preconditions and effects, for each step we encode explicitly the state of the system *before* and *after* an action is performed; we do so by appending A and B respectively to the variable names. Thus, if x is a variable describing the state of a component then xA_j and xB_j encode the component state before and after the jth action.

Actions. Each action has a unique integer identifier. For $j \in \{1, \ldots, n\}$ we use

- A_j to store the identifier of the action performed in step j,
- t_j is the time when the execution of the action of step j starts and
- rd_j is the time needed to complete the action of step j.

Robots. The identity and state of the robot executing the action of step $j \in \{1, \ldots, n\}$ will be described using the following variables:

- R_j stores the integer identifier of the robot executing step j,
- $holdA_j$ and $holdB_j$ tell whether the robot is holding something before respectively after the action at step j and

- pos_j specifies the position where the robot needs to be to execute the action assigned at step j.

Machines. The identity and state of the machine used in step $j \in \{1, \ldots, n\}$ is encoded by the following integer-valued variables:

- M_j tells what machine is involved in the action performed at step j,
- md_j specifies the action duration,
- $state1A_j$ and $state1B_j$ encode whether the machine used in step j is prepared before resp. after the step,
- $state2A_j$ and $state2B_j$ encode whether a CS used at step j is loaded with a cap or not and
- $state3A_j$ and $state3B_j$ encode whether the slide of a CS used in step j is full or not.

Initial state. We introduce dedicated variables to describe the initial state. Though the game always starts in a fixed initial state, such variables give us the flexibility to synthesize plans on-the-fly during the game. We define for all $i \in \{1, ..., R\}$ and $k \in \{1, \ldots, M\}$:

- $initPos_i$ and $initHold_i$ to encode initial conditions for robot i and
- $initState1_k$, $initState2_k$ and $initState3_k$ to store the initial state for machine k.

Rewards. To define the objective function to be optimized, we use for each step $j \in \{1, \ldots, p\}$

- a real-valued variable rew_j to store the reward achieved for executing step j.

Using the above variables, we define the encoding of plans as shown in Figure 3.8. In the following, *products* are encoded by integer values – e.g., "no product" is represented by 0, black base by 1, etc. We start with defining subformulas to encode the initial system state, the preconditions and effects of the possible actions and the rewards that can be achieved.

Initialization. For the initial state of the game we define the φ_{init} in formula (3.17), meaning that robots start from the depot and do not hold objects, while machines are not prepared nor loaded for production.

Making initial states consistent. φ_{start} in formula (3.18) ensures that the above initial values are propagated to the initial states for robots and machines. If robot i is active at step j and it has never been active before, then j is its first step and it must start in the robot's initial state. Moreover, for each step, the robot timer is incremented by *at least* the travel time, which is encoded using constants $Dist(u, q)$ for the travel time between the machines u and q. Similar requirements are imposed on the machines.

Making successor states inductively consistent. The formula φ_{id} in formula (3.19) ensures that when a robot or machine is not involved in an action then the action does not change the robot's (resp. machine's) state. The formula states that if robot i is active at step j' and it has not been active since step $j < j'$, then we ensure that its *hold* state is propagated to j' (we say that effects of previous step j are equal to the preconditions at j'). The robot moves to the location required by the action assigned at j'. The robot timer will be incremented by *at least* travel time plus action duration. A similar interpretation holds for the machines.

Action rules. formula (3.20) defines φ_a that specifies the preconditions and effects of action a. The formula means that when an action a – encoded by its integer identifier – is selected, the appropriate preconditions are checked and effects are propagated. For instance, the rule for the delivery action will have the following definition:

$$A_j = 11 \implies (M_j = 2 \wedge state1A_j = 8 \wedge state1B_j = 0 \wedge$$
$$state2B_j = state2A_j \wedge state3B_j = state3A_j \wedge md_j{=}15 \wedge$$
$$pos_j = 2 \wedge holdA_j = 3 \wedge holdB_j = 0 \wedge rd_j = 10)$$

Reward scheme. Finally, we need to specify a reward scheme for actions. As already mentioned, by means of rewards we can drive the synthesis towards optimal plans. We chose to assign positive rewards to the delivery action only,

while all other actions bring no rewards. The reward is defined in formula (3.21) by φ_{rew} where dl is the deadline for delivering a specific product and $t_j + md_j$ indicates the instant when the appropriate station completes the delivery process. Such reward strategy yields plans that minimize the makespan of the plan executed by robots.

Plans. Optimal plans can now be encoded by the problem to maximize rew_p under the side condition $\varphi_{init} \wedge \varphi_{start} \wedge \varphi_{id} \wedge (\bigwedge_{a \in A} \varphi_a) \wedge \varphi_{rew}$, where A is the set of all actions needed to produce the requested product.

$$\varphi_{init} := \left\{ \begin{array}{l} \bigwedge_{i=1}^{R} initHold_i = 0 \wedge initPos_i = 0 \wedge \\ \bigwedge_{k=1}^{M} initState1_k = 0 \ \wedge \ initState2_k = 0 \wedge initState3_k = 0 \end{array} \right. \tag{3.17}$$

$$\varphi_{start} := \left\{ \begin{array}{l} \bigwedge_{i=1}^{R} \bigwedge_{j=1}^{n} (R_j = i \wedge \bigwedge_{j'=1}^{j-1} \neg(R_{j'} = i) \implies (holdA_j = initHold_i) \wedge \\ \bigvee_{u=0}^{M} \bigvee_{q=1}^{M} initPos_i = u \wedge pos_j = q \wedge t_j \geq Dist(u,q)) \wedge \\ \bigwedge_{k=1}^{M} \bigwedge_{j=1}^{n} (M_j = k \wedge \bigwedge_{j'=1}^{j-1} \neg(M_{j'} = k) \implies (state1A_j = initState1_k \\ \wedge \ state2A_j = initState2_k \wedge state3A_j = initState3_k)) \end{array} \right. \tag{3.18}$$

$$\varphi_{id} := \left\{ \begin{array}{l} \bigwedge_{j=1}^{n} \bigwedge_{j'=j+1}^{n} (R_{j'} = R_j \wedge \bigwedge_{j''=j+1}^{j'-1} \neg(R_{j''} = R_j) \implies (holdA_{j'} = holdB_j) \\ \wedge \bigvee_{u=0}^{M} \bigvee_{q=1}^{M} pos_j = u \wedge pos_{j'} = q \wedge t_{j'} \geq t_j + rd_j + Dist(u,q)) \wedge \\ \bigwedge_{j=1}^{n} \bigwedge_{j'=j+1}^{n} (M_{j'} = M_j \wedge \bigwedge_{j''=j+1}^{j'-1} \neg(M_{j''} = M_j) \implies \\ (state1A_{j'} = state1B_j \wedge \ state2A_{j'} = state2B_j \\ \wedge state3A_{j'} = state3B_j \wedge t_{j'} \geq t_j + md_j)) \end{array} \right. \tag{3.19}$$

$$\varphi_a := \left\{ \begin{array}{l} A_j = id \implies (preconditions \wedge effects) \end{array} \right. \tag{3.20}$$

$$\varphi_{rew} := \left\{ \begin{array}{l} rew_j = dl - t_j - md_j \end{array} \right. \tag{3.21}$$

Figure 3.8: SMT encoding for the transition system underlying the RCLL domain.

3.4.2 Experimental evaluation

To evaluate plans computed by our system, we consider the production process shown in Figure 3.9. Assembling a C_0 product requires the following actions:

ID	Action
1	Retrieve base with cap from shelf at CS
2	Prepare CS to retrieve cap
3	Feed base into CS
8	Discard cap-less base
7	Prepare BS to provide black base
6	Retrieve base from BS
4	Prepare CS to mount cap
5	Feed black base to CS
9	Retrieve black base with cap from CS
10	Prepare DS for slide specified in order
11	Deliver to DS

We generated 100 problems, determined by a unique machine placement and order set each. This allows for qualitatively validating plan generation and determining costs of plans generated. We vary the complexity through the number of robots participating in the task. In the following, we report an experimental analysis that focuses on encodings for producing a single product of the lowest complexity C_0 (Figure 3.9).

We compare our solutions with domains encoded in PDDL2.1 [FL03]. We consider both, temporal domains with durative actions[6] (T) and the same domains without (NT).

[6]PDDL2.1 allows for more expressive temporal reasoning than the one we support in our encoding, e.g., timed initial literals, effects happening at the beginning or at the end of an action. Here we limit ourselves to the case where effects are can only happen at start and last for the whole duration of an action.

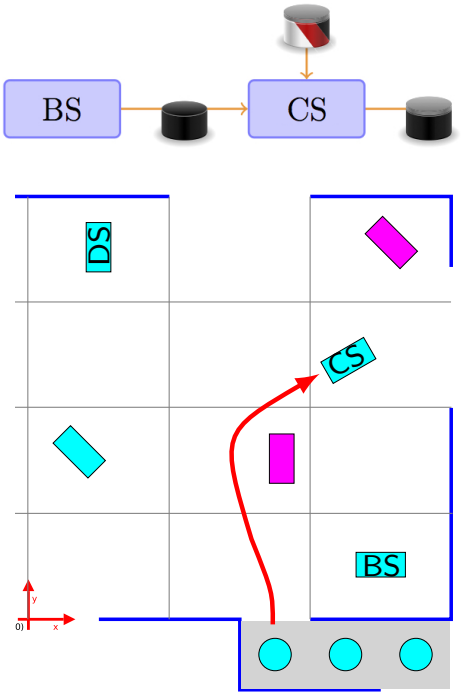


Figure 3.9: Production of a C_0 product (top). The first step for the robot is to move to feed a cap in to the Cap station (bottom). [NLF15, RCL17].

We run planners and solvers on a machine running Debian 9, Intel Core 2 Quad CPU Q9450 at 2.66 GHz and use the benchmark files generated using the simulation. We also validate results generated by our approach using the simulator developed for the Planning Competition for Logistics Robots in Simulation shown in Figure 3.1.

Evaluation of OMT solvers

We started with a comparison of performances of three OMT solvers, namely, νZ, SMT-RAT and OptiMathsat on this benchmark. A timeout for solving is set to 60 seconds, which is the time teams can afford spending in planning during an RCLL game without compromising their chances to win. Solving for the domain considered proved to be challenging, however plans could be successfully synthesized with our approach. νZ was the only tool which could solve all the instances proposed, therefore it was chosen for our analysis.

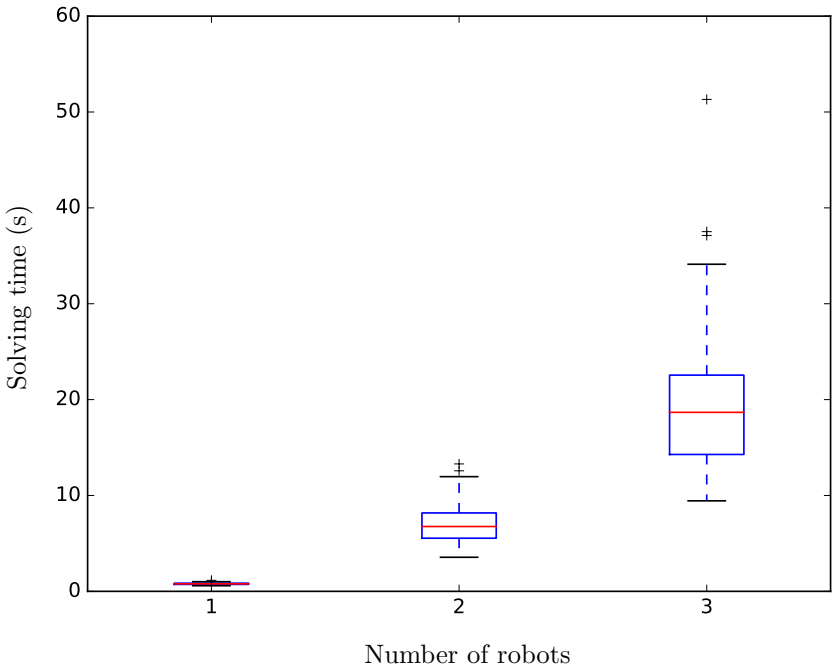


Figure 3.10: Boxplots for solving times using νZ for 1, 2, 3-robot teams. Red lines represent median values of $0.78s$, $6.76s$ and $18.67s$ respectively.

We investigated how the solving time varies with the number of robots used. As Figure 3.10 shows, the solving time increases with the size of the team of robots used, moving from an average solving time of $0.79s$ with only one robot, to $19.45s$ for three robots. A natural explanation for this could be that having more robots increases the size of the search space and therefore the number of solutions, which are equivalent up to renaming but do not improve the quality of the plan. Still, the solver will perform an exhaustive check when computing an optimal plan and this results in a harder solving process. In any case, the times obtained are well within the suggested desirable limits for the RCLL competition.

Off-line comparison with other approaches

In the off-line comparison, we consider the POPF [CCCG11] planner and SMT-Plan [CFLM16], a tool that compiles PDDL domains into SMT encodings and solves them by calling νZ internally. We choose the former as it comes readily integrated with ROSPlan [CFL+15], a framework for task planning and execu-

	One robot					Two robots					Three robots				
	OMT	POPF/NT		POPF/T		OMT	POPF/NT		POPF/T		OMT	POPF/NT		POPF/T	
	o	a	o	a	o	o	a	o	a	o	o	a	o	a	o
# of instances solved	**100**	**100**	**100**	**100**	**100**	**100**	0	0	19	19	**100**	0	0	9	9
solving time average (s)	**0.79**	5.09	0.87	20.73	11.02	**7.06**	–	–	25.66	17.68	**19.45**	–	–	34.25	31.30
plan makespan average (s)	**64.1**	186.99	99.22	67.49	76.06	**51.98**	–	–	60.09	62.10	**51.85**	–	–	54.65	57.56

Table 3.3: Comparison of OMT and POPF for temporal (T) and non-temporal (NT) domains using anytime (a) and one-shot (o) planning.

tion we used in the validation presented in the following. SMTPlan, instead, was selected because it represents an interesting solution building a bridge between AI planning and SMT solving. Both tools are evaluated on non-temporal (NT) and temporal (T) domains.

Table 3.3 shows the results of this comparison, carried out using a timeout of 60 seconds, a typical time still acceptable in the RCLL. A total of 100 different domain instances were run for each approach for 1, 2 and 3 robots respectively, resulting in a total of 900 runs. For POPFanytime we report the time needed for the planner to compute an improved solution, although the tool still ran for the whole 60 seconds allocated. SMTPlan is not listed, as it timed out for all the instances considered. We conjecture that this may be due to the way PDDL domains are compiled to SMT, resulting in unnecessarily redundant encodings that are difficult to solve. We can observe in Table 3.3 that only OMT could solve all benchmarks within the given timeout. While POPF could always compute solutions for domains where only one robot was used, it failed to do so when the number of robots increased. Furthermore, our approach is able to solve the synthesis problem in less time, when the comparison is possible, and produces solutions with average makespans that are always smaller than other approaches.[7] Furthermore, giving POPF additional time to optimize on

[7]Makespan for non-temporal POPF with single robot is computed as follows. We read the sequence of actions contained in the plan and assign to each the same duration specified in the temporal models used by other approaches.

the first feasible solution (anytime) did not seem to lead to major improvements compared to the one-shot evaluation. We should mention that all models (OMT and PDDL-based) use approximate values to represent action durations. In particular we assume for the navigation actions that the robot moves at $1\,{}^{\mathrm{m}}/_{\mathrm{sec}}$, i.e., using distance as time. While this is unrealistic for actual execution, the values remain comparable among the approaches.

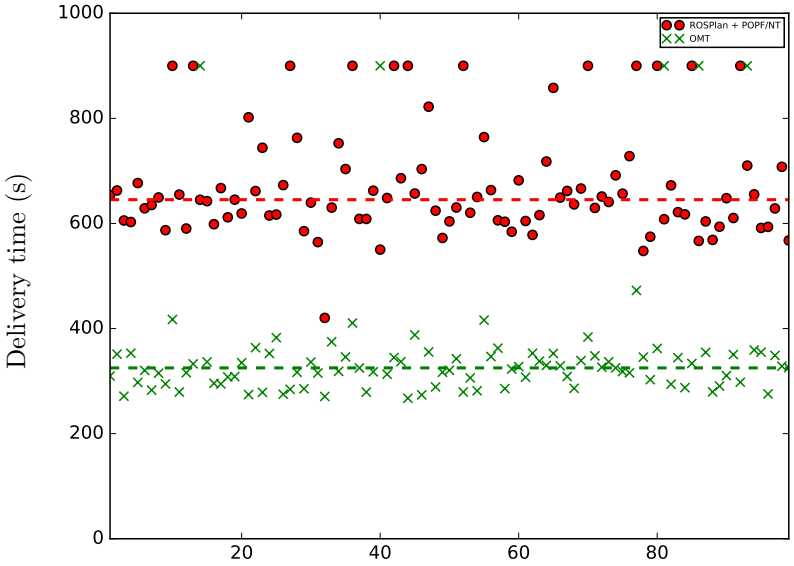
Validation of results

Plans generated with our approach were validated in the Planning Competition for Logistics Robots in Simulation using the framework described in Section 3.2.

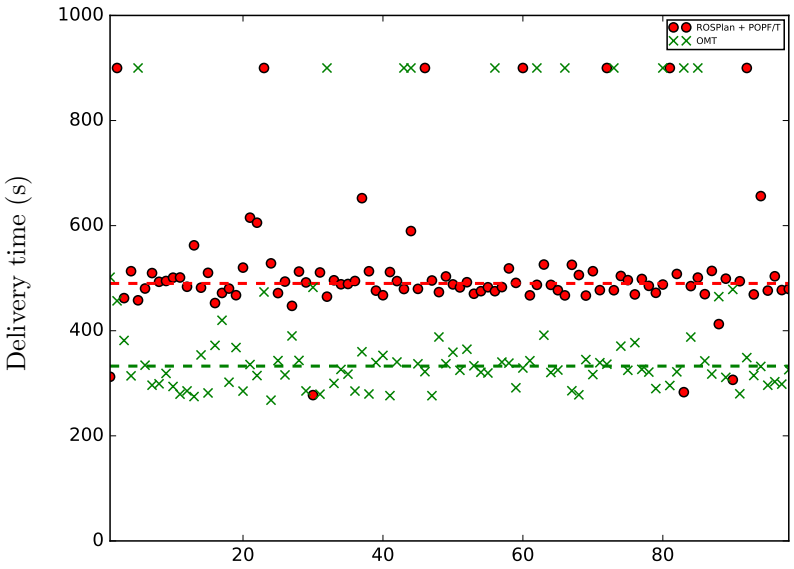
We tested the robustness of our solutions under realistic competition settings by having two teams of robots competing against each other, one being controlled with our approach. Had we tested using one team of robots only (that is, our team), we would have reduced the uncertainty present in the game due to the strategies adopted by the opponent. To control the other team, we considered two approaches: *(i)* a PDDL-based approach that embeds `POPF` into ROSPlan and, *(ii)* a purely rule-based approach based on CLIPS [NLF13], currently used by the RCLL world champions. It must be noted that the execution engine currently used in our framework supports concurrent execution of actions on multiple robots, ROSPlan does not.

We therefore decided start our experimental campaign with plans synthesized for single robots, and have them compete with ROSPlan using a single robot. Figure 3.11 shows statistics for 100 simulations, where our approach competed with ROSPlan combined with non-temporal (a) and temporal (b) reasoning. We plot delivery times for both approaches and for each game.

Confirming our off-line results, our plans were able to control the robot to deliver the order requested. However, for some simulations, plans computed by OMT or ROSPlan failed to be executed – we set the corresponding delivery time to 900 seconds. For what concerns our approach, we suggest this may be due to the fact that we assume all machines in the shop-floor are correctly working, however sometimes machines are out of order for a limited time to simulate real world failures. Since we do not capture this uncertainty in our logical encoding, it may happen that the assumptions about the world state



(a)



(b)

Figure 3.11: Game statistics for a single robot, OMT playing against ROSPlan combined with POPF using non-temporal (a) and temporal (b) reasoning (20 seconds timeout).

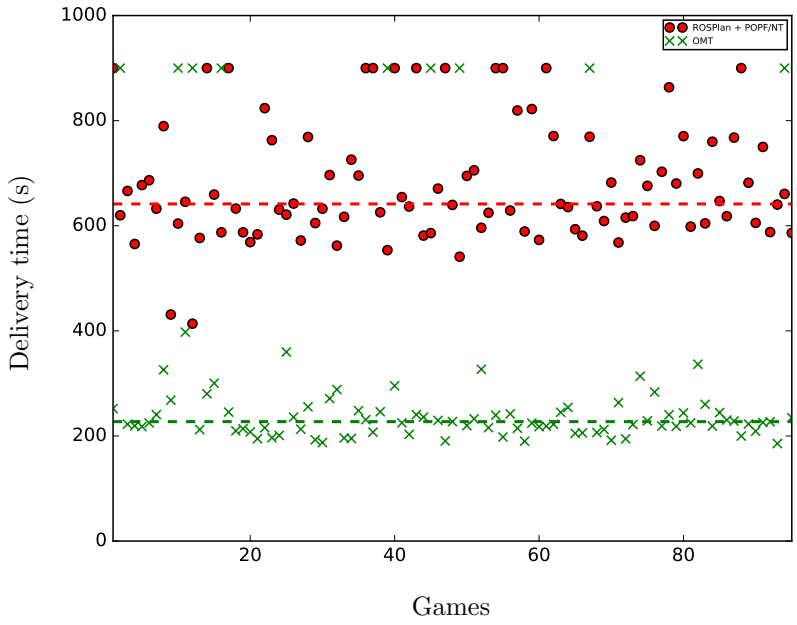


Figure 3.12: Game statistics for OMT plans (multi-robot) *versus* ROSPlan combined with POPF using non-temporal reasoning (single robot) (timeout 20 seconds).

made during synthesis become inconsistent during execution. During the first batch of games (Figure 3.11, a) we can observe that our plans failed 5 times, while the opponent failed 12 times. In all other cases we could deliver products successfully within the deadline of the game (15 minutes). Comparing delivery times between the two approaches would not be fair in this case, as ROSPlan did not perform any temporal reasoning during these games. We therefore proceeded with a comparison with ROSPlan combined with temporal reasoning (Figure 3.11, b). There, our approach failed 11 times, while ROSPlan failed 7. However, we can observe that when successful, our team had a median delivery time of $\sim 332\,\mathrm{s}$, against $\sim 490\,\mathrm{s}$ of the other team. Such simulations reflect the results we obtained during our off-line evaluation, where our approach could compute plans with the smallest makespans.

We then proceeded with the evaluation of plans synthesized for multiple robots. Synthesizing global plans for multi-robot teams could, in principle, increase the chances of failure due to,e.g., synchronization issues. To test the robustness of our plans, we ran 100 games where ROSPlan (again, single robot)

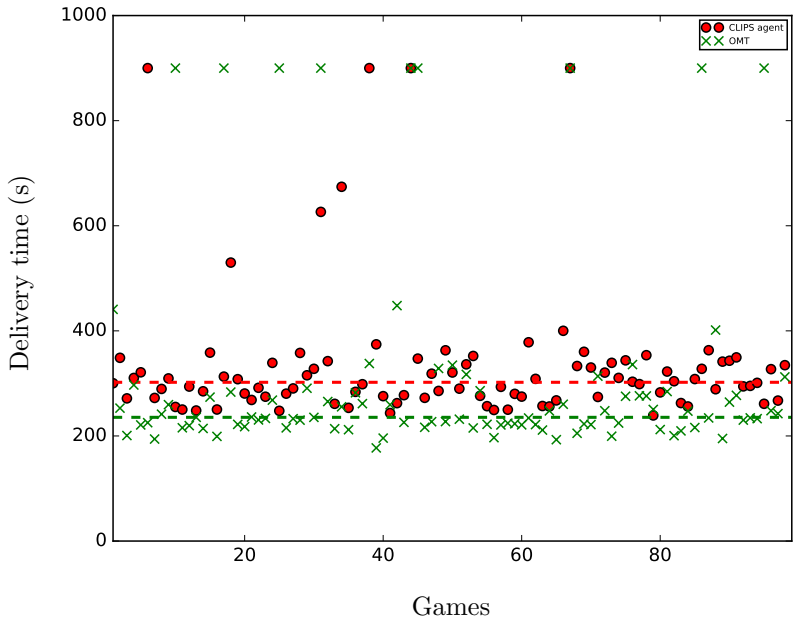


Figure 3.13: Game statistics for OMT (multi-robot, timeout 20 seconds) playing against the rule-based approach presented in [NLF13] (multi-robot).

competed against our approach, where multiple robots were used.

Figure 3.12 shows results obtained after 100 games. Interestingly our plans proved to be as robust as serial plans computed for a single robot. Indeed, our approach failed 9 times while ROSPlan failed 12. Given that our approach employed multiple robots, median delivery times for our team are always lower than the opponent's.

Finally, we compared the performances of our plans with the rule-based approach used by the RCLL world champions. This approach employs the full team of robot, allowing a fair comparison between solutions for multi-robot systems. Figure 3.13 shows the results obtained after 100 games. Results obtained show that, when successful in delivering, our approach guarantees a shorter delivery time, with a median delivery time of $\sim 235\,\mathrm{s}$ against $\sim 302\,\mathrm{s}$ of the rule-based approach. On the other hand, the rule-based agent proved to be more robust, failing only 4 times against 9 times of our approach.

3.5 Explaining plans

The problem of generating explanations for decisions taken by autonomous robotic systems is a very pressing one. The effectiveness of these systems is limited by their current inability to explain their decisions and actions in a human-readable way. Several initiatives have been launched recently to tackle this problem. For instance, DARPA started the *Explainable AI program*[8] with the aim to develop new machine-learning techniques that will produce more explainable models that could be translated into understandable and useful explanation dialogues for the end user. In the same spirit, *Explainable Planning* is proposed in [FLM17], where the authors consider the opportunities that arise in AI planning to form a familiar and common basis for communication with the users.

In this section we discuss how OMT-based synthesis implemented in our system could be leveraged to generate explanations for the plan synthesis process. While we acknowledge the existence of a gap between the way OMT solving proceeds and human problem-solving, here we wish to show that OMT solvers exploit techniques that have the potential to ease explaining and facilitate understanding of the underlying decision process.

In particular, we discuss explanations that can be used to understand *(i.)* why a certain plan should be preferred, or *(ii.)* why no plans could be produced for a given scenario. The discussion that follows is intended to provide initial ideas for achieving the objective of providing effective explanations in OMT synthesis. Examples discussed are specific to the RCLL domain, however we expect that our results can provide a basis for general synthesis of explanations supporting OMT-based decision making.

Explaining why a plan should be chosen. The first question we wish to consider is explaining why a solution computed by the solver should be preferred over different ones. To the best of our knowledge, there exists no planner able to optimize for a metric different than minimizing plan makespan. Therefore, while answering such a question could prove challenging, if not impossible, in other AI-based solutions, OMT could provide useful answers.

[8] https://www.darpa.mil/attachments/DARPA-BAA-16-53.pdf

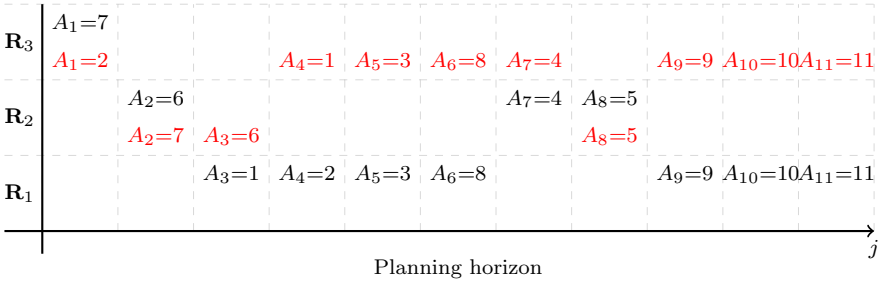


Figure 3.14: Example of plans with minimum makespan for the production of a C_0 piece, using two (red) and three (black) robots. The makespan of both plans is 58.52s . Action are encoded by integer ids as in Figure 3.9.

As introduced in Section 2, OMT differs from SMT solving in that it produces solutions that are not only *feasible*, but also *optimal*. The key point here is that OMT allows to specify different metrics to measure the quality of a plan. Modern OMT solvers support combinations (lexicographic, pareto, box) of objective functions that can be specified by the user. In this framework, a valid explanation could be to point out the differences in the metrics and show the different effects they have, e.g., in terms of the obtained final reward.

Example. Let us consider a simple example based on the production process of Figure 3.9. Let us assume a plan has been requested by the user and the reward scheme of formula (3.21) – which yields plans with minimum makespan – has been used. A sample plan as produced by one of our plans might have the structure depicted in black in Figure 3.14, where three robots are used.

Now suppose we want to know whether a better makespan could be achieved using less robots. One simple way to check this could be to extend our optimization problem by including an additional objective, e.g.,

$$maximize \sum_{j=1}^{p} R_j$$

which implicitly forces the solver to select robots whose integer ids have higher value – e.g., robot 3 will be preferred over robot 1. The result is shown in red in Figure 3.14. As we can see, it is sufficient to ask the same robot – robot 2 in this case – to perform actions 7 and 6 to obtain a plan that has the same makespan as the original one, but uses only two robots. So in this case, by pointing out

the differences in the metrics used to drive the synthesis procedure, one could produce a reason as to explain to the end-user why the second solution should be preferred over the first one.

Explaining why a plan can not be synthesized. This question arises when the solver fails to synthesize a plan for the problem at hand. Search-based planners are typically not very effective at proving unsolvability of a plan. In contrast, OMT-based approaches are well positioned to address this challenge. Our system frames plan synthesis as a bounded model-checking problem, therefore if the solver states that the desired objective can not be met within a given deadline (and/or within a planning horizon) then this is a proof that no plan can be produced to accomplish the task.

Besides proving the non-existence of a plan, modern OMT solvers also allow to extract *unsatisfiable cores* that additionally provide a reason for unsatisfiability. Formally, given an unsatisfiable input formula $\varphi = \wedge_{i=1}^{n} \varphi_i$, an unsatisfiable core of φ is an unsatisfiable formula $\psi = \wedge_{i \in I} \varphi_i$ for some $I \subseteq \{1, \ldots, n\}$. In other words, an unsatisfiable core of φ is an unsatisfiable formula ψ which is either φ itself or $\varphi = \psi \wedge \psi'$ for some ψ'.

Though smaller unsat cores typically provide more compact information, *minimal* unsat cores (i.e., unsat cores $\wedge_{i \in I} \varphi_i$ for which $\wedge_{i \in I'} \varphi_i$ is satisfiable for all $I' \subset I$) are computationally hard to compute. Therefore, most solvers aim at generating small explanations but they seldomly guarantee minimality. Since for practical problems unsat cores might be too large to be analyzed by humans, SMT/OMT solvers that follow the SMT-LIB standard[9] require that the user specifies a label for each of the conjunctive subformulas (also called *assertions*) of interest, and only the labeled formulas in the unsat core are listed as output (i.e., the provided explanation together with the unlabeled assertions form an unsat core).

Example. To illustrate how unsat cores can be used to explain unsolvability, consider the following example. The RCLL rules impose that machines can be out of service for a given time at any point in the game. To capture this information, we extend the encoding of machine states of Section 3.4 by introducing the

[9] https://smtlib.github.io/jSMTLIB/SMTLIBTutorial.pdf

integer-valued variables $state0A_j$ and $state0B_j$. Such variables encode whether the machine used in step j is fully functional before and after action A_j respectively. If a machine goes down, then all the actions involving that machine can not be performed any more, making it impossible to complete the production of pieces requiring the broken machine. To model this, we extend each action rule (formula (3.20), Section 3.4) with the additional precondition that the machine required at step j must be working. For instance, the rule for the delivery action will become:

$$
\begin{aligned}
&A_j = 11 \implies \\
&(M_j = 2 \wedge state0A_j = 1 \wedge state0B_j = 1 \wedge \\
&state1A_j = 8 \wedge state1B_j = 0 \wedge state2B_j = state2A_j \wedge \\
&state3B_j = state3A_j \wedge md_j = 15 \wedge \\
&pos_j = 2 \wedge holdA_j = 3 \wedge holdB_j = 0 \wedge rd_j = 10)
\end{aligned}
$$

Furthermore, we label each constraint in order to enable unsat core generation. Let us now assume our synthesis procedure is triggered under the condition that the *delivery station* DS is broken. This means that the actions involving DS won't be realizable, as a precondition for them to be performed is that the machine has to be operational, i.e., $state0A_j = 1$.

```
1    Start solving...
2    No solution found
3    Time: 0.209315061569
4    Unsat core:
5    prepare_DS_for_slide_specified_order_10
```

Listing 3.5: Unsat core generated when DS is down.

Listing 3.5 shows the unsat core produced by νZ when we impose that the delivery station breaks at step 10, i.e., $state0A_{10} = 0$. The unsat core produced here shows that the delivery station could not be prepared for delivery, therefore making delivery impossible.

Chapter 4

Planning with OMT: a general approach

Chapter 3 presented our first, domain-specific, attempt to solve planning problems using OMT technology. Although promising, the results obtained did not allow for a more general assessment of the applicability of OMT to broader classes of numeric problems studied by the planning community. To make up for this, we initially focused our efforts on generalizing the concept of long-distance dependencies to domain-independent planning. Unfortunately, this generalization did not produce the hoped-for results and did not lead to significant advancements to the state of the art in numeric planning. We therefore decided to explore a different direction from our previous work and developed a new, general OMT-based approach to cost-optimal numeric planning based on abstraction refinement. With this new encoding we could show that OMT-based approaches have an edge on other solutions when dealing with some interesting classes of numeric problems. This chapter is meant to provide a formal characterization of our construction.

4.1 Expressiveness versus tractability

Planning for realistic problems requires expressive languages to model the world. These languages, however, must achieve two conflicting goals: on the one hand,

they need to be expressive enough so that faithful problem representations are possible; on the other hand, they need to be simple enough so that general, efficient algorithms can be developed to solve the problems represented.

The existence of this trade-off between expressiveness and tractability, however, does not mean that progress is not possible. Indeed, methods that can deal effectively with (some) interesting problems represented in decidable fragments of first-order theories have been proposed. In this chapter we focus on one such extension of classical planning that relies on arithmetic theories, that is, *numeric planning*. We consider the challenge of solving to optimality problems in this class, and advance the state of the art with a new planning approach that can handle a fragment of numeric problems for which (integrated) algorithmic solutions have not been proposed yet.

As introduced in Chapter 2, numeric planning [FL03] is an extension of classical planning where state variables can be numeric and actions' preconditions and effects may involve arithmetic reasoning over such variables. Despite being undecidable in the general case [Hel02], notable advances have been achieved for restricted fragments of numeric planning. Admissible heuristics have been extended to handle *simple* numeric planning problems [SHT16] in which actions have linear conditions and may only increase or decrease numeric variables by a constant – see, e.g, [SHT16, SHMT17, PCCB18b]. Several heuristics have been proposed for numeric planning problems where both conditions and effects are expressed as *linear* expressions over numeric state variables [Hof03, IM17, LSHB18]. However these are non-admissible and therefore cannot be used in the cost-optimal setting. Cost-optimal planning with both simple and linear effects can be handled by [PCCB18a] via a compilation to MILP that proved to be competitive with heuristic search approaches.

In this chapter we take a step further and extend cost-optimal numeric planning to problems where conditions may be simple or linear *and* actions are equipped with *state-dependent* costs, i.e., costs are encoded by linear expressions over numeric state variables. Previous works have studied state-dependent action costs (SDAC) [IHT$^+$14] in the classical setting [GKM15, GKM16] and in the presence of global numerical state constraints [IHT$^+$14, IGH19], but did not explore extensions towards numeric planning.

Support and scalability are challenging in the setting we target. Complex numeric structures require a formalism that is expressive enough to capture them; at the same time, the added expressiveness comes at the price of poorer scalability. We believe that recent advances in satisfiability checking could be leveraged to address these challenges. First we present a novel SMT encoding of numeric planning that enables a relaxed reachability analysis that does not require building long, intractable formulas. In a nutshell, standard planning formulas are extended with a fixed-length suffix that performs a boolean abstraction of the transition relation. Reasoning on this abstraction, we can conclude whether a goal is reachable only with a modest increase in the size of the formula and without resorting to expensive decision procedures for theory-reasoning. We then discuss how this construction can be extended to enable optimal planning using Optimization Modulo Theories (OMT) [ST15a], an extension of SMT that combines efficient propositional reasoning with dedicated procedures for theory-optimization. In order to implement the above, we extend well-known techniques for compiling planning into SAT [KS96, RHN06, Rin12] and SMT [SD05, CFLM16]. We leverage these results to build what, to the best of our knowledge, is the first domain-independent theory-planner based on OMT and use it to test our scheme. We measure its performance over several domains previously reported in the literature [SHMT17, LSHB18], and on a new numeric domain featuring SDAC which we introduce below. This domain, although not modeling a real application, represents a good example of a problem which state-of-the-art tools struggle to solve.

4.1.1 Running example

To illustrate the workings of our approach, we introduce a new planning domain called Security Clearance. In this domain, an intelligence agency has to manage clearance authorizations for several documents across different security levels. The agency can authorize a level to read a document, however doing so changes the clearance of the document: authorizations at lower levels are revoked, while those at higher levels remain unchanged. Authorizing a level has a cost which directly corresponds to the level involved, e.g., authorizing level 2 costs 2. Since some documents may be more important than others, each

```
(define (problem security-clearance-1-2)

(:domain security-clearance)

(:init
(not (clear_d1_l1) )
(not (clear_d1_l2) )
(= (cost_d1) 0)
(= (priority_d1) 1)
(= (high) 3)
(= (low) 1)
)

(:goal (and
(clear_d1_l1 )
(clear_d1_l2 ))
)

(:metric minimize  (cost_d1))

)
```

Listing 4.1: PDDL representation of the initial state, goal state and plan quality metric for the Security Clearance domain with one document d_1 and two levels l_1, l_2.

document is initially assigned a priority. When needed, the agency can increase it incurring in a cost proportional to the current priority of the document. If a document has high priority, the agency can decide to authorize all levels at once by paying the appropriate price. Starting from an initial situation where no level is authorized, the goal for the agency is to authorize all levels to read all documents while minimizing expenses. Listings 4.1 and 4.2 show a PDDL model for the Security Clearance domain with one document d_1 and two levels l_1, l_2.

4.2 Optimal planning modulo theories

In the following we describe a new encoding for numeric planning with OMT, which we build in two steps. For a given horizon n, we first extend standard state-based encodings with one additional step T^R (from n to $n+1$, see Figure 4.1). This step performs a boolean abstraction of the effects of each action:

```
(define (domain security-clearance)

(:predicates
(clear_d1_l1)
(clear_d1_l2)
)

(:functions
(cost_d1)
(priority_d1)
(high)
(low)
)

(:action increase_priority_d1
:parameters ( )
:precondition (and (< (priority_d1) (high)) )
:effect (and (increase (priority_d1) 1)
(increase (cost_d1) (priority_d1)))
)

(:action authorize_all_d1
:parameters ( )
:precondition (and (>= (priority_d1) (high))
(not (clear_d1_l1))
(not (clear_d1_l2)) )
:effect (and (clear_d1_l1) (clear_d1_l2)
(increase (cost_d1) 2))
)

(:action authorize_d1_l1
:parameters ( )
:precondition (and (not (clear_d1_l1)))
:effect (and (clear_d1_l1) (increase (cost_d1) 1))
)

(:action authorize_d1_l2
:parameters ( )
:precondition (and (not (clear_d1_l2)))
:effect (and (clear_d1_l2) (not (clear_d1_l1))
(increase (cost_d1) 2))
)
)
```

Listing 4.2: PDDL domain for the Security Clearance domain with one document d_1 and two levels l_1, l_2.

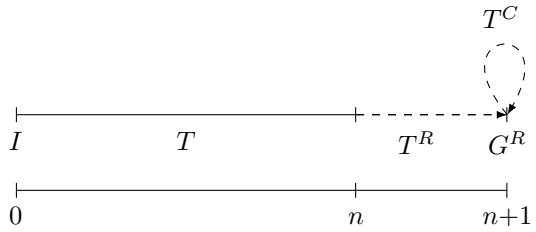


Figure 4.1: High-level representation of the new construction for planning formulas.

actions executed in n still enforce their preconditions on concrete variables (i.e., variables at n), however their effects are replaced with fresh boolean variables indicating that the corresponding concrete variables may have changed their value. Informally, this abstraction stores information about portions of the state space that can still be modified by actions executed after n. Since abstract effects may in turn trigger the (abstract) execution of other actions, the second step of our construction computes the transitive closure of all facts that are true in $n + 1$ under an abstract transition relation enforcing both preconditions and effects on variables in the abstract space (T^C in Figure 4.1). In so doing, we obtain all variables whose value could still be changed if a longer horizon was given. The intuition behind this abstraction is as follows. If the goal formula is not yet satisfied in n and its variables cannot be modified any further, the corresponding planning formula is unsatisfiable and the problem does not admit any solution. Notably, this abstraction can also be leveraged to perform optimal reasoning, as we will describe in the second part of this section.

Since the new encoding is somewhat involved, we use the SECURITY CLEARANCE domain to exemplify each the building block of our construction. For the sake of readability, however, we present the construction considering a single action, *authorize_d1_l1*, which we simply rename to *authorize* – a similar renaming is applied to state fluents.

Building the boolean abstraction. For a given horizon n, we enforce initial condition and transitions until n as per formula (2.5.1) without any change. When step n is reached, we append one additional step. As previously introduced, this step performs a *boolean abstraction* of the transition relation and

transfers the computation to an abstract space where original state variables do not appear anymore. To exemplify this step, consider the aforementioned action *authorize*. The *abstraction-enabling execution* of this action is modeled by a new action $t\text{-}authorize$ where: *(i)* the new action has the same precondition of the original action and, *(ii)* the new action enforces a boolean abstraction through its effects. Using this construction, we specify the semantics of an execution of $t\text{-}authorize$ from step n to $n{+}1$ as

$$\begin{aligned} t\text{-}authorize_n &\Rightarrow \neg clear_n \\ t\text{-}authorize_n &\Rightarrow t\text{-}clear_{n+1} \\ t\text{-}authorize_n &\Rightarrow t\text{-}cost_{n+1} \end{aligned} \tag{4.1}$$

where the first axiom ensures that $t\text{-}authorize$ is executed in n only if enabled, and the remaining enforce the booleanization of the effects of $t\text{-}authorize$ via auxiliary variables $t\text{-}clear, t\text{-}cost \in \mathbb{B}$. Notice that mutex axioms are not included as these abstract transitions only keep track of changes that may happen, disregarding the actual concrete effect.

While axioms (4.1) are sufficient to model the execution of $t\text{-}authorize$, some important information is still missing from the abstract space. Indeed, we must make sure that knowledge about facts that are already satisfied at step n is transfered to the abstract space at $n{+}1$. This means we must compute the boolean abstraction of those facts that hold in n and are not abstracted by the effects of actions executed in n. We therefore introduce fictitious actions that achieve this goal. In our example,[1] we would introduce an action $t\text{-}sat^{\neg clear}$ with the following semantics:

$$\begin{aligned} t\text{-}sat^{\neg clear}_{n} &\Rightarrow \neg clear_n \\ t\text{-}sat^{\neg clear}_{n} &\Rightarrow t\text{-}con^{\neg clear}_{n+1} \\ t\text{-}con^{\neg clear}_{n+1} &\Rightarrow t\text{-}sat^{\neg clear}_{n} \end{aligned} \tag{4.2}$$

where $t\text{-}sat^{\neg clear}_{n}$ can be executed only if $\neg clear_n$ holds, and knowledge about this fact is transfered at step $n{+}1$ via an auxiliary variable $t\text{-}con^{\neg clear}_{n+1} \in \mathbb{B}$

[1] Notice that in this simplified version of SecurityClearance no additional actions would be needed as the abstraction already contains all the relevant information. Clearly, this may not always be the case.

which can only be set by $t\text{-}sat^{\neg clear}_{n}$. This construction requires the introduction of one $t\text{-}sat$ action for each atomic precondition appearing in ground actions. In the following, we use TA to denote the set of abstract action variables (i.e., the set of t-actions) and TV to denote the set of abstract state variables (i.e., the set of t-variables and $t\text{-}con$'s). Furthermore, let $T^R(\mathcal{V}, TA, TV)$ represent the conjunction of all abstraction-enabling actions (including fictitious actions).

Computing the minimum reachable set. When step $n{+}1$ is reached, the booleanization is complete. However, as a result of this abstraction, the execution of new actions may be enabled in the abstract space. In order to understand whether executing these actions would positively contribute towards reaching the goal, we need to compute all facts that are reachable in the abstract space.

To enable this computation, we further abstract $T^R(\mathcal{V}, TA, TV)$ by asserting preconditions on abstract variables only. In our example, we abstract $t\text{-}authorize$ to $tt\text{-}authorize$ with the following execution semantics

$$\begin{aligned}
tt\text{-}authorize_{n+1} &\Rightarrow t\text{-}con^{\neg clear}_{n+1} \vee t\text{-}clear_{n+1} \\
tt\text{-}authorize_{n+1} &\Rightarrow t\text{-}clear_{n+1} \\
tt\text{-}authorize_{n+1} &\Rightarrow t\text{-}cost_{n+1} \\
t\text{-}clear_{n+1} &\Rightarrow t\text{-}authorize_n \vee tt\text{-}authorize_{n+1} \\
t\text{-}cost_{n+1} &\Rightarrow t\text{-}authorize_n \vee tt\text{-}authorize_{n+1}
\end{aligned} \tag{4.3}$$

where the first axiom asserts the new preconditions, the second and third enforce boolean effects as done previously and the last specifies frame conditions on auxiliary variables.

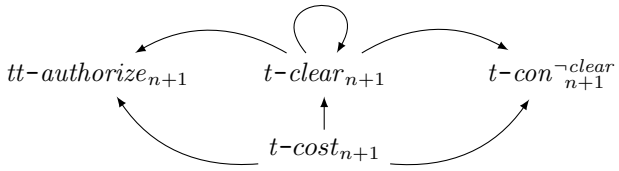
Intuitively, the new preconditions assert that abstract actions can be executed if either the concrete precondition was already true at step n or, its abstract counterpart can be modified at step $n + 1$. Since this computation is flattened in one layer, we need to make sure the corresponding assignments are valid. In other words, given all facts that are true in $n{+}1$ we must compute their *smallest closure* under the abstract transition relation to obtain all valid reachable abstract states. Drawing from the field of Logic Programming, we notice that this corresponds exactly to computing the *answer set* [GL88] of the

rules corresponding to actions executed at step $n{+}1$.[2] In our example we have:

$$\frac{tt\text{-}authorize_{n+1},\ t\text{-}con^{\neg clear}_{n+1}}{t\text{-}clear_{n+1},\ t\text{-}cost_{n+1}} \qquad \frac{tt\text{-}authorize_{n+1},\ t\text{-}clear_{n+1}}{t\text{-}clear_{n+1},\ t\text{-}cost_{n+1}} \tag{4.4}$$

Previous works on compilation from Answer Set Programming [GL88] to SAT [LZ02] or SMT [Nie08] can be leveraged to encode this computation in our construction. We use *loop formulas* as introduced in [LZ02] and adapt them to the planning setting.

We start by constructing a *dependency graph* for actions at step $n{+}1$ as a directed graph D such that: *(i.)* D has a node for each variable appearing in action rules and *(ii.)* for each rule there is an edge connecting variables appearing in the conclusion to variables in the premises. For instance, the dependency graph for rules (4.4) is



From [LZ02] we know that strongly connected components of D represent loops in the formula, i.e., circular dependencies that hinder the computation of the smallest set closed under action rules. To remove loop L at step $n{+}1$, we add the formula

$$\bigvee L \Rightarrow \bigvee R(L) \tag{4.5}$$

where L is the set of variables representing the loop, $\bigvee L$ denotes a disjunction over all elements of L and the set $R(L)$ is built by adding, for each rule whose conclusions intersect L, those premises that do not intersect L. In our running example we have one loop $L = \{t\text{-}clear_{n+1}\}$ and the corresponding loop formula writes

[2] This idea bears some similarities to the relaxed reachability analysis of [Hel09], although applied in a different setting and with a different construction.

$$t\text{-}clear_{n+1} \Rightarrow t\text{-}con^{\neg clear}_{n+1} \wedge tt\text{-}authorize_{n+1} \tag{4.6}$$

In the following we use TTA to denote the set of all tt-action variables and write $T^C(TTA, TV)$ to denote the conjunction of all abstract actions and related loop formulas.

A new planning formula. We are now left to check whether goal states are reachable in the abstract space. To model this, we abstract goal axioms using the same notion of abstraction seen for preconditions in axioms (4.1), e.g., goal condition $clear_n$ becomes $t\text{-}con^{clear}_{n+1} \vee t\text{-}clear_{n+1}$. We use G^R to denote the formula obtained after abstracting G. For an horizon n, a new planning formula Π_{n+1} is constructed as

$$\begin{aligned} &I(\mathcal{V}_0) \wedge \bigwedge_{i=0}^{n-1} T(\mathcal{V}_i, A_i, \mathcal{V}_{i+1}) \wedge T^R(\mathcal{V}_n, TA_n, TV_{n+1}) \\ &\wedge T^C(TTA_{n+1}, TV_{n+1}) \wedge G^R(TV_{n+1}) \end{aligned} \tag{4.7}$$

With the construction we just introduced, we can formulate the following results.

Theorem 1. *(unsolvability) For any $n \geq 0$, if Π_{n+1} is unsatisfiable, the planning problem Π does not admit solution.*

Proof sketch. Assume Π_{n+1} is unsatisfiable while Π admits a plan. Let $\pi_m = \langle A_0, \ldots, A_{m-1} \rangle$ be the solution of Π at horizon m. If $m \leq n$ then Π_{n+1} is satisfiable by construction therefore assume $m > n$. Again, by construction the sequence $\langle A_0, \ldots, A_{n-1} \rangle$ satisfies Π_{n+1}. Abstraction-enabling axioms ensure that A_n is still applicable in Π_{n+1} and its effects enable the abstract execution of subsequent sets $\langle A_{n+1}, ..., A_{m-1} \rangle$. Since A_{m-1} reaches the goal in Π, the abstract goal can be reached in Π_{n+1} as well, leading to a contradiction. □

4.2.1 Extension to OMT

We now show how to extend the SMT encoding to enable optimal planning. Formulating problems with costs – be they constant or state-dependent – in numeric PDDL is straightforward: objective functions can be expressed by any

arithmetic expression and actions can have arbitrarily complex effects on numeric variables.

OMT can handle this added expressiveness, hence the OMT encoding has the same structure of formula (4.7), with the following notable changes.

First, a metric for plan quality is added to the encoding. This extension requires no effort as OMT solvers accept the specification of optimization metrics via dedicated facilities offered by SMT-LIB [BFT16], the standard language of SMT/OMT.

Consequently, booleanization of effects on cost variables is removed from T^R and T^C. In our abstraction, actions enforce costs defined as follows. Actions in T^R are assigned the same cost of their concrete counterparts at step n ; actions in T^C are assigned their minimum cost over all states. Actions introduced with axioms (4.2) instead have zero cost, as performing any of them should not affect plan quality. Mutex axioms on cost variables are not added as costs are directly enforced by pseudo-boolean terms added to the optimization metric. In our example, we would assert cost 1 for both $t\text{-}authorize_n$ and $tt\text{-}authorize_{n+1}$ (no state-dependency) and 0 for $t\text{-}sat^{\neg clear}_n$. The objective function to be minimized would write

$$cost_n + 1 * t\text{-}authorize_n + 1 * tt\text{-}authorize_{n+1}$$

Next, we turn our attention to the construction of loop formulas. These formulas play a fundamental role in ensuring the soundness of Theorem 1. However they are not suited for optimal reasoning, at least in their current form. To see this, consider the following scenario. Assume action $t\text{-}authorize$ is executed at step n. From axioms (4.1), $t\text{-}clear_{n+1}$ is set to true, which, in turn, enables axiom (4.6). While $t\text{-}con^{\neg clear}_{n+1}$ is taken care of by axioms (4.2), the loop formula forces the execution of $tt\text{-}authorize_{n+1}$, although not needed. While this does not affect reachability results, it clearly affects reasoning about costs ($tt\text{-}authorize_{n+1}$ has non-zero cost). To alleviate this problem we remove tt-action variables from the construction of the dependency graph. This is allowed only because these variables never appear in conclusions of action rules, hence can safely be disregarded. As a result of this, loop formulas change, and for instance formula (4.6) becomes

$$t\text{-}clear_{n+1} \Rightarrow \neg clear_n \tag{4.8}$$

With the extensions above, we need to make sure the solver does not push the execution of all actions to the suffix, where they would have minimum cost. Not only this would affect optimal reasoning, but would also affect termination of bounded planning procedures. Indeed we would need to increase the planning horizon indefinitely hoping to find a valid plan (i.e., containing only concrete actions), but this would never happen. Hence to ensure termination, we augment the OMT encoding with the following axioms. For each action $a \in A$ let $M_a \subseteq A$ be the set of actions that are not independent from a. For each action $a \in A$ and for each step $0 < i < n$ we add

$$a_i \Rightarrow \left[a_{i-1} \vee \left(\bigvee_{\varphi \in pre_a} \neg \varphi_{i-1} \right) \vee \left(\bigvee_{a' \in M_A} a'_{i-1} \right) \right] \tag{4.9}$$

With axioms (4.9) an action a is taken at step i only if: *(i.)* a was already performed at step $i-1$ or, *(ii.)* a was not applicable at step $i-1$ or, *(iii.)* another action a', mutex with a, was performed at $i-1$. These same axioms are also enforced on abstract actions at steps n and $n+1$, although on actions in TA and TTA.

Let Π^+_{n+1} denote the planning formula extended with the axioms above, the following result holds.

Proposition 1. *For any $n \geq 0$, Π_{n+1} and Π^+_{n+1} are equisatisfiable.*

With the addition above we can formulate the following theorem.

Theorem 2. *(optimality) For any $n \geq 0$, let μ be the optimal solution of Π^+_{n+1}. If $\mu \models G_n$ then μ is a valid* optimal *plan.*

Proof sketch. The proof is based on the fact that the goal state could be reached without resorting to abstract actions. Since the cost of these actions is always less than or equal to the cost of all other actions in Π, we see that adding any more actions to μ would inevitably lead to solutions with higher cost. □

Algorithm 1 Optimal Planning Modulo Theories

```
1:  procedure OMTPlan(Π, ub)
2:      set initial horizon n := 0
3:      while n ≤ ub do
4:          build formula Π+_{n+1}
5:          if Π+_{n+1} is UNSAT then
6:              return Π does not admit solution;
7:          else
8:              extract model μ of Π+_{n+1} ;
9:              if μ satisfies G_n then
10:                 return μ
11:             else
12:                 increase horizon n;
13:     return no plan found within bound ub
```

Planning algorithm We embed the construction presented in the previous sections into a new planning procedure, which we call OMTPlan – see Algorithm 1. Given a planning problem Π and an upper bound ub, our procedure builds bounded encodings for increasing horizons (lines $2-4$). At each iteration, we check if formula Π^+_{n+1} is not satisfiable. If that is the case, the procedure terminates according to Theorem 1 and signals that the planning problem does not admit a solution (lines $5-6$). If formula Π^+_{n+1} is satisfiable instead, we extract a model μ for it in line 8. Notice that μ has minimum cost among all possible models of Π^+_{n+1}, being the result of an OMT check. We then check the condition expressed in Theorem 2 and, depending on the result, we either return the optimal plan represented by μ or increment the horizon for the next iteration (lines $9-12$). Finally, if no solution can be found within the given upper bound, the procedure terminates signaling failure in line 13.

4.3 Empirical evaluation

To evaluate our planning procedure OMTPlan, we developed a prototypical implementation in Python 2. Our implementation leverages the modules devel-

oped in [EMR09] for parsing, and uses the Python API[3] of νZ [BPF15] to build and solve OMT formulas.

Implementation of the encoding We implemented an optimized version of the encoding presented above. Indeed, upon inspection of axioms (4.1,4.2,4.3) we can observe that variables $t\text{-}con$'s and $t\text{-}sat$'s can be removed by variable elimination. Applying this step results in the following, more compact, formulation[4]

$$\begin{aligned} t\text{-}authorize_n &\Rightarrow \neg clear_n \\ t\text{-}authorize_n &\Rightarrow t\text{-}clear_{n+1} \\ tt\text{-}authorize_{n+1} &\Rightarrow \neg clear_n \vee t\text{-}clear_{n+1} \\ tt\text{-}authorize_{n+1} &\Rightarrow t\text{-}clear_{n+1} \\ t\text{-}clear_{n+1} &\Rightarrow t\text{-}authorize_n \vee tt\text{-}authorize_{n+1} \end{aligned} \tag{4.10}$$

where preconditions once enforced on $t\text{-}con$'s are now directly enforced on state variables at step n. After eliminating $t\text{-}con$'s and $t\text{-}sat$'s, loop formulas change accordingly, i.e., each variable $t\text{-}con$ appearing in a loop formula is replaced with the corresponding precondition at step n.

Analysis and discussion Our experimental analysis compares with search based approaches implemented in the ENHSP planner [SHTR16] and with the MILP compilation (C^{SC}) of [PCCB18a]. Experiments are carried out using a 30 minute timeout and 4 GB memory limits on a machine running Debian 3.16 with processor Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz.

Our analysis considers numeric problems with simple and linear conditions, and also numeric domains with state-dependent action costs. Simple numeric domains are taken from [SHMT17]; linear domains are from [LSHB18], with two additions, Rover-Metric and FO-Zenotravel, developed starting from their simple counterpart. Finally, we generate planning problems with SDACs using the Security Clearance domain we introduced in previous sections.

[3] `https://github.com/Z3Prover/z3/wiki/Documentation`

[4] Booleanization of cost variables is omitted and relaxed actions enforce costs as explained in the previous section.

Domain	#	$\hat{h}^{rmax}$		C^{SC}		OMTPlan		Best
		C	T	C	T	C	T	
Counters	15	6	28.22	**15**	1.36	7	524.59	15
Depots	20	**3**	1050.02	1	4.9	1	78.48	3
Farmland	30	**30**	193.68	28	32.86	1	211.57	30
Gardening	63	**63**	599.85	**63**	887.33	18	3031.23	63
Sailing	20	16	2101.13	**17**	2813.55	5	345.23	17
Satellite	20	2	293.1	**4**	459.8	1	17.85	4
Rover	20	**4**	25.91	**4**	10.93	**4**	61.5	4
Zenotravel	20	6	579.3	**7**	699.65	4	107.74	7
Total	213	130	4871.21	**139**	4910.38	31	4378.19	143

Table 4.1: Coverage (C) and total solving time (T) in seconds for domains with simple conditions.

For domains with simple effects we compare against the $\hat{h}^{rmax}$ heuristic of [SHMT17] and the MILP compilation (C^{SC}) of [PCCB18a]. Table 4.1 shows coverage and the total solving time. Results confirm the efficiency of C^{SC} on simple numeric problems, outperforming other approaches on almost all instances. On the other hand, our approach suffers from two main drawbacks. The first one is that domains like Farmland, Gardening and Sailing feature optimal plans with relatively many steps and little parallelism. Such "long and narrow" plans force us to produce large encodings that exceed the capabilities of μZ before finding optimal solutions. The second drawback has to do with our choice of axioms (4.9) in the encoding, and can affect our performance adversely even in domains, e.g., Counters, where optimal plans are "short and wide", i.e., featuring relatively few steps and lots of parallelism. Indeed, while (4.9) tries to make sure that actions are taken before entering the suffix, it may still happen that the optimal solution for a fixed horizon is a relaxed solution which also satisfies (4.9). In such cases, we are still forced to increment the horizon until we exceed the capability of the underlying solver. Note that the interaction between relaxation and axioms (4.9) is not always harmful as the adverse effect depends on the structure of the domain and the associated costs.

In domains with linear effects we compare our encodings with C^{SC} and with

Domain	#	h^{blind}		C^{SC}		OMTPlan		Best
		C	T	C	T	C	T	
FO-Count	20	4	339.84	3	223.83	**9**	2104.71	9
FO-Count-Inv	20	3	77.29	2	48.82	**6**	937.41	6
FO-Count-Rnd	60	14	1411.79	10	520.29	**23**	2835.46	23
FO-Farmland	50	**13**	1035.09	2	47.07	1	6.21	13
FO-Sailing	20	**2**	610.85	0	-	1	71.56	2
Rover-Metric (1-10)	10	4	151.69	4	14.02	**5**	303.39	5
TPP-Metric (1-10)	10	**5**	20.51	n.a.	n.a.	3	524.12	5
Zenotravel-Linear	20	**4**	145.5	2	1.55	**4**	888.24	4
Total	220	49	3792,29	23	855.58	**52**	7671.10	67

Table 4.2: Coverage (C) and total solving time (T) in seconds for domains with linear conditions. Entries reporting n.a. indicate that the planner could not be run on the corresponding domain.

a weighted A^* search using a simple goal sensitive heuristic (h^{blind}) that returns 0 if the state is a goal state and 1 otherwise. Table 4.2 reports results obtained in linear domains. Solving 52 problems, OMT outperforms other approaches, still leaving room for improvement. On some domains, OMTPlan manages to outperform the competitors. We believe that this is due to the increased complexity in the numerical part which OMTPlan handles comparatively better than the other approaches. However, OMTPlan is still challenged by domains like Tpp-Metric, a variant of the Traveling Salesman Problem with no parallelism.

We finally turn our attention to Security Clearance. We generated 36 instances of the domain, varying the number of documents (from 2 to 10) and the number of levels (from 2 to 5). Exploring this domain both in depth and breadth, we can investigate weaknesses of constraint and search-based methods respectively. Here, C^{SC} cannot be considered for our analysis as it does not provide support for state-dependent cost structures. Hence, we compare only with h^{blind}. Figure 4.2 shows a cactus plot of the result obtained. As one can observe the domain proved to be challenging for both approaches, with OMT being able to solve 26 instances and h^{blind} solving 16. The performance of h^{blind} degrades when the number of documents is increased, incurring in what could be

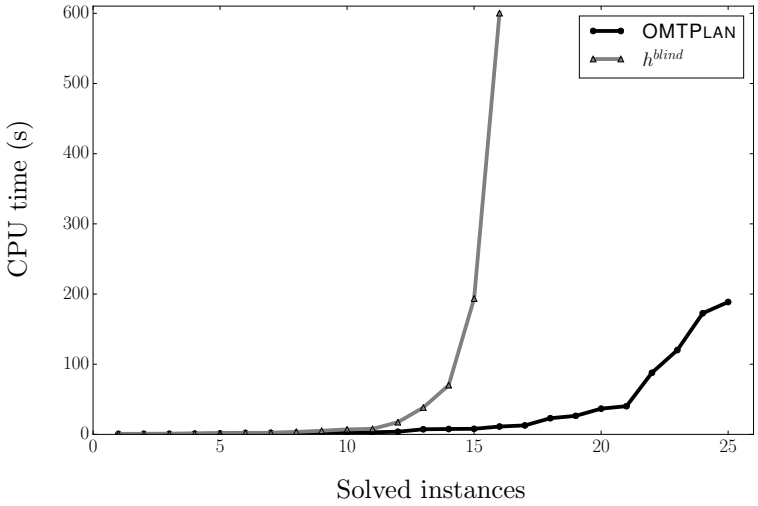


Figure 4.2: Cactus plot for the SECURITY CLEARANCE domain. Instances are ordered by increasing CPU time, reported in seconds.

explained as a worst-case behavior of A^*. Indeed, the planner produces timeouts for almost all instances having strictly more than 5 documents, while already failing to solve some problem with less documents. OMTPlan's performance is comparable to h^{blind} for instances with up to 4 documents (all levels), while a considerable difference can be noticed for instances with higher number of documents. In particular, OMTPlan always manages to solve instances with 2 or 3 levels, even in domains with 10 documents. Still, domains having 4 or 5 levels proved challenging and could not be solved for instances having 6 documents or more.

4.4 Conclusion

We considered the problem of generating optimal plans for numeric domains with costs that can be either unitary, constant or state-dependent. Since solving these problems require an efficient interplay between propositional and arithmetic reasoners, we proposed Optimization Modulo Theories as the framework of choice. We presented a novel encoding of planning problems that enables efficient reasoning about optimality by abstraction. We provided a characterization of this abstraction, as well as a practical planning algorithm that uses it. We further provided empirical evidence of the usefulness of our approach, demon-

strating state-of-the-art results on some expressive classes of numeric problems.

Chapter 5

The OMTPlan planner

From a practical perspective, one of the most important outcomes of the work carried out during the elaboration of this thesis is the OMTPlan planner, where the ideas discussed in Chapter 4 have been implemented and tested. In this short chapter we discuss the architecture of the planner and related implementation details, while some future development lines are discussed in Chapter 6.

The planner is open-sourced under a GNU General Public License, version 3 (GPL-3.0), and available for download at the following link:

`https://github.com/fraleo/OMTPlan`

Different parts of the planner make use or are built on top of the following third-party software components:

- Parsing of PDDL files and grounding are done using a modified version of the Python parsing component of the Temporal Fast-Downward planner [EMR09], available at `http://gki.informatik.uni-freiburg.de/tools/tfd`;
- SMT and OMT formulas are built and solved using the Python API of the νZ solver [BPF15], now part of the Z3 suite available at `https://github.com/Z3Prover/z3`;
- Plans produced by OMTPlan are validated using the plan validator VAL, available at `https://github.com/KCL-Planning/VAL`.

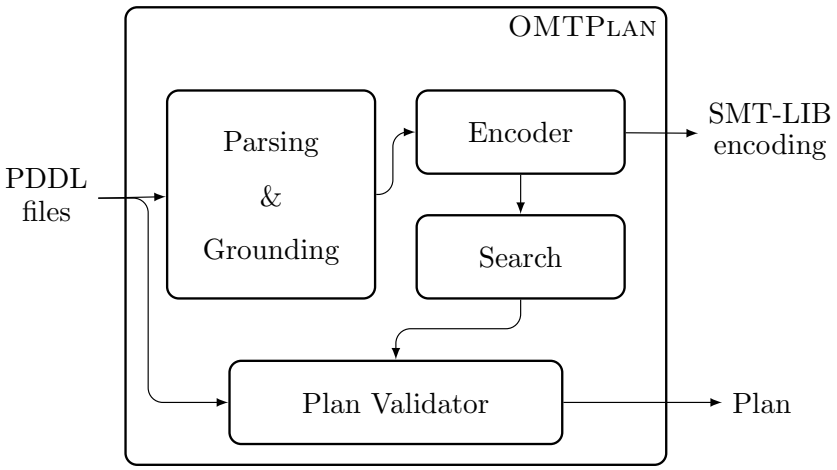


Figure 5.1: Schematic representation of the architecture of OMTPlan.

5.1 System description

The OMTPlan planner realizes its functionalities through the interaction of several components that are represented in Figure 5.1. Each component takes care of different phases of the planning process as detailed in the following.

Parsing and grounding: PDDL files containing the description of the planning problem at hand are fed to the planner and are parsed by this module. To this end we leverage Python parsing module developed for the Temporal Fast-Downward planner. This module required some minor modifications to be adapted to our architecture, with one notable exception. Indeed, he original implementation of the parsing module does not provide support for the specification of arbitrary metric for plan quality. Hence, we had to apply some changes to make sure the parser would behave appropriately when confronted with metrics supported by our approach.

Besides parsing operations, this module is also responsible for grounding the first-order representation used in PDDL. The grounding algorithm used makes use of a compilation to a logic program in order to perform reachability analysis and grounding all in one [Hel09]. This compilation is particular to both the set of action schemas and the initial state of the search. The reachability analysis is able to infer if certain ground actions will never be applicable when starting from the given initial state, and that some fluents will never actually change

their value, i.e., they are seen as static facts. As a result, ground actions that are not applicable are pruned (and not encoded in the SMT formula) and static fluents are compiled away and do not need to be represented with SMT variables in the planning formula.

Encoder: Once parsing and grounding operations have been performed, an instance of this module is created. Upon creation, the encoder is fed the parse tree as returned by the previous module. The main task of this module is to traverse the parse tree and build the corresponding planning formulas. In the current implementation the user can choose between three different types of encodings:

- SMT encodings for satisficing planning;
- SMT encodings for optimal planning (unit costs[1]);
- OMT encodings for optimal planning (constant costs and SDAC).

The first two encodings are classical, meaning that they use the standard state-based representation seen in the Planning as SAT literature, here extended to numeric variables. The main difference between the two lies in the encoding of the execution semantics: the first allows for parallel actions, while the second only allows for serial plans. The reason for this is simple: optimal plans for problems with unit costs are plans with the minimum number of actions. A fairly naive strategy to compute such plans is to allow only one action per step and build formulas increasing the planning horizon one by one until the first solution is found. This solution also corresponds to the global optimum for the planning problem at hand. This restriction is lifted in the third encoding. OMT encodings are built as described in Chapter 4 and can handle both serial and parallel executions as will always return the optimal solution for the given metric.

Finally, this module is also responsible for exporting SMT-LIB encodings [BFT16] of planning formulas built at different horizons. This functionality serves two different purposes:

[1]This is the simple case where all actions have the same cost, and this equals one.

- *debugging*: human-readable SMT-LIB encodings can be used to detect bugs in the logic of the encoder;
- *benchmarking*: the SMT community can leverage the extensive benchmark suite developed by the planning community to test strengths and weaknesses of solvers on numeric planning formulas. Benchmarks can be generated for both SMT and OMT.

Search: This module is responsible for implementing horizon allocation strategies to be adopted during plan search. The current implementation supports a standard *ramp-up* strategy adapted to the encoding chosen by the user. In the case of satisficing SMT planning and optimal OMT planning, we use an exponential progression of horizons until a user-provided upper bound is reached. The optimal SMT encoding leverages the strategy described above.

Once a search strategy has been selected, this module schedules calls to the encoder to produce planning formulas for different horizons. Horizons are tested sequentially, although other strategies, such as [Rin12], could easily be implemented. The search module is also responsible for feeding the planning formula to the underlying solver, fetching the result of the satisfiability check and act according to such result. The current implementation relies, for this step, on νZ [BPF15], however other OMT solvers such a OptiMathSAT [ST15b] could be used.

Validation: When a plan is found by the underlying solver, the validation module is called to decide whether the plan is indeed correct. Besides receiving the plan as input, this module also requires the PDDL files containing the description of the planning problem (domain and problem files). The validation task is the performed by the plan validator VAL, which checks whether the plan computed complies with the PDDL definition of the problem. If a plan is deemed valid, it is passed on to the main routine for subsequent operations, otherwise OMTPlan reports failure.

Chapter 6

Conclusion

6.1 Summary of contributions

Let us conclude this thesis with a summary of our contributions and a discussion of potential lines of research that might follow this work.

The first of these contributions (Chapter 3) is a domain-specific approach to solve planning problems arising from the RoboCup Logistics League. We presented several OMT encodings for both phases of the RCLL and evaluated their applicability within an integrated system for planning, execution and monitoring. To the best of our knowledge, this system represents the first use of Optimization Modulo Theories in planning. Notably, this very system participated, and won, the Planning and Execution Competition for Logistics Robots in Simulation held at the International Conference on Automated Planning and Scheduling (ICAPS) in 2018.

The second contribution (Chapter 4) is the development of a new algorithm for domain-independent (optimal) planning as OMT that allows to handle numeric domains with state-dependent action costs. The algorithm relies on a new construction of planning formulas, which leverages abstraction to answer reachability and optimality questions without incurring in the blow-up that would typically occur with standard constructions. We presented a formal characterization of this new encoding and evaluated it empirically on well-known, as well as, new benchmarks for numeric planning. This algorithm represents the first

attempt to use Optimization Modulo Theories technology to solve expressive numeric planning problems.

The third and last contribution (Chapter 5) is the implementation of OMT-PLAN, the first OMT-based planner ever proposed. The planner implements the ideas discussed in Chapter 4 and leverages know-how gained while working on the encodings presented in Chapter 3. The current implementation of OMT-PLAN supports both satisficing planning with SMT and optimal planning with OMT.

6.2 Open challenges and future work

As a result of the efforts put into solving the problem presented in this work, we gained interesting insights on the problem of planning as OMT. We detail in the following some observations and ideas that could be useful for other researchers considering venturing in this field.

Problem-specific knowledge. Incorporating problem-specific knowledge in the solving process can lead to considerable speed-ups. Solvers specifically tuned to handle planning problems have been already proposed in the context of Planning as SAT [GMS98, Rin11], but this direction has never been explored in the SMT and OMT framework.

Solvers and optimization. There exist efficient solvers for different types of optimization problems like combinatorial optimization or integer programming. However, there seems to be room for improvements on problems where the objective function is an arithmetic function but the search is over a finite set of objects, i.e., where the problem seems to involve optimization in the arithmetic domain but at its core it is a purely combinatorial optimization problem. For instance, in the RoboCup Logistics League, the plan generation problem could be specified as a Boolean combination of *equalities* between arithmetic terms, i.e., only combinatorial optimization plays a role. However, the solvers do not recognize this fact and invoke also arithmetic optimization. For the latter, equalities seem to be more problematic, therefore we partially replaced them by inequalities and forced equalities by the objective function. This is an example

where knowledge about the internal solving mechanisms is needed to achieve better encodings.

Parallelization. Practical efficiency is probably one of the main limitations of current OMT algorithms and tools. Beyond theoretical worst-case complexity results, research on SAT and SMT has shown that problem instances arising from application domains can be tackled successfully in many cases of interest. The research on OMT is currently less mature than its SMT counterpart, therefore improving on this aspect is still an open challenge. We believe that parallelization may offer an interesting path towards attaining practical efficiency in OMT. While parallel SAT solving has been the subject of research in the past [HW13], the development of parallel SMT solvers is still in its infancy [HW18], and, to the best of our knowledge, paradigms for parallel OMT have not been considered yet. Carrying over the results obtained in SAT to SMT/OMT is nontrivial, because the role of SAT solver inside SMT/OMT procedure is to *enumerate* assignments and not just to *search* a satisfying one. However, once enumeration of assignments can be successfully parallelized, at least to some extent, also checking their theory consistency and, possibly, finding optimal solutions, can be distributed on several processors. Besides modern multi-core architectures, parallelization could also take advantage of hybrid CPU-GPU architectures, where the GPU part can substantially speed up numerical computations as it happens in other AI fields like training of deep neural networks — see, e.g., [KSH17].

Develop new encodings. Our experiments with different encodings of planning problems into OMT indicate that considerable progress can be made by considering novel kinds of encodings and relaxations. Beyond computational concerns, new relaxations can be of great interest from a representational standpoint. One key challenge relates to finding encodings which *generalize* well to several problem domains. The work presented in Chapter 4 represents a first step in this direction, although other forms of planning relaxations, such as the interval-based relaxation of [SHTR16], could be tested in conjunction with OMT.

Planning with datatypes. The satisfiability checking techniques that we used in our work can be applied more or less off-the-shelf not only to arithmetic theories, but also to structured data types such as intervals, bit-vectors or arrays. Satisfiability Modulo Theories solvers do indeed support reasoning over such types, and extending our approach to support them too should not be a challenge. Indeed, our algorithm should be able to support planning over any background theory with little additional effort, as long as the basic semantics of the theory is supported by solvers. This could be seen both as a straight-forward implementation of the Planning Modulo Theories framework of [GLFB12] and its extension to optimal planning.

Bibliography

[BFT16] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[BG01] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33, 2001.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[BLPT19] Arthur Bit-Monnot, Francesco Leofante, Luca Pulina, and Armando Tacchella. SMT-based planning for robots in smart factories. In *Proc. of IEA/AIE*, pages 674–686, 2019.

[BN95] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–656, 1995.

[BPF15] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νz - An optimizing SMT solver. In *Proc. of TACAS*, pages 194–199, 2015.

[BSST09] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. IOS Press, 2009.

[Byl94] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[CCCG11] Andrew Coles, Amanda Jane Coles, Allan Clark, and Stephen Gilmore. Cost-sensitive concurrent planning under duration un-

certainty for service-level agreements. In *Proc. of ICAPS*, pages 34–41, 2011.

[CFG+10] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Roberto Sebastiani, and Cristian Stenico. Satisfiability modulo the theory of costs: foundations and applications. In *Proc. of TACAS*, pages 99–113, 2010.

[CFL+15] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcís Palomeras, Natàlia Hurtós, and Marc Carreras. Rosplan: planning in the robot operating system. In *Proc. of ICAPS*, pages 333–341, 2015.

[CFLM16] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. A compilation of the full PDDL+ language into SMT. In *Proc. of ICAPS*, pages 79–87, 2016.

[CGSS13] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The Mathsat5 SMT solver. In *Proc. of TACAS*, pages 93–107, 2013.

[CKJ+15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. In *Proc. of SAT*, pages 360–368, 2015.

[Dan02] George B. Dantzig. Linear programming. *Operations Research*, 50(1):42–47, 2002.

[DK01] Minh B. Do and Subbarao Kambhampati. Sapa: a domain-independent heuristic metric temporal planner. In *Proc. of ECP*, pages 109–120, 2001.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proc. of TACAS*, pages 337–340, 2008.

[DN13] Carmel Domshlak and Anton Nazarenko. The complexity of optimal monotonic planning: the bad, the good, and the causal graph. *J. Artif. Intell. Res.*, 48:783–812, 2013.

[DNK97] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in nonmonotonic logic programs. In *Proc. of ECP*, pages 169–181, 1997.

[EMR09] Patrick Eyerich, Robert Mattmüller, and Gabriele Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proc. of ICAPS*, pages 130–137, 2009.

[ER99] Stefan Edelkamp and Frank Reffel. *Deterministic state space planning with BDDs*. Citeseer, 1999. Available at `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.9126`.

[FL03] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

[FL06] Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res.*, 27:235–297, 2006.

[FLM17] Maria Fox, Derek Long, and Daniele Magazzeni. Explainable planning. *CoRR*, abs/1709.10256, 2017.

[FN71] Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971.

[For82] Charles L. Forgy. Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, September 1982.

[GB13] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

[Gia07] Joseph C. Giarratano. *CLIPS Reference Manual*, 2007. Available at `http://clipsrules.sf.net/OnlineDocs.html`.

[GKM15] Florian Geißer, Thomas Keller, and Robert Mattmüller. Delete relaxations for planning with state-dependent action costs. In *Proc. of IJCAI*, pages 1573–1579, 2015.

[GKM16] Florian Geißer, Thomas Keller, and Robert Mattmüller. Abstractions for planning with state-dependent action costs. In *Proc. of ICAPS*, pages 140–148, 2016.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP*, pages 1070–1080, 1988.

[GLCT19] Dario Guidotti, Francesco Leofante, Claudio Castellini, and Armando Tacchella. Repairing learned controllers with convex optimization: a case study. In *Proc. of CPAIOR*, pages 364–373, 2019.

[GLFB12] Peter Gregory, Derek Long, Maria Fox, and J. Christopher Beck. Planning modulo theories: extending the planning paradigm. In *Proc. of ICAPS*, pages 65–73, 2012.

[GLTC19] Dario Guidotti, Francesco Leofante, Armando Tacchella, and Claudio Castellini. Improving reliability of myocontrol using formal verification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(4):564–571, 2019.

[GMS98] Enrico Giunchiglia, Alessandro Massarotto, and Roberto Sebastiani. Act, and the rest will follow: exploiting determinism in planning as satisfiability. In *Proc. of AAAI*, pages 948–953, 1998.

[Hel02] Malte Helmert. Decidability and undecidability results for planning with numerical state variables. In *Proc. of AIPS*, pages 44–53, 2002.

[Hel09] Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5-6):503–535, 2009.

[HG01] Patrik Haslum and Héctor Geffner. Heuristic planning with time and resources. In *Proc. of ECP*, pages 121–132, 2001.

[HNCL16] Till Hofmann, Tim Niemueller, Jens Claßen, and Gerhard Lakemeyer. Continual planning in Golog. In *Proc. of AAAI*, pages 3346–3353, 2016.

[Hof01] Jörg Hoffmann. FF: the fast-forward planning system. *AI Magazine*, 22(3):57–62, 2001.

[Hof03] Jörg Hoffmann. The metric-ff planning system: translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res.*, 20:291–341, 2003.

[HW13] Youssef Hamadi and Christoph M. Wintersteiger. Seven challenges in parallel SAT solving. *AI Magazine*, 34(2):99–106, 2013.

[HW18] Antti E. J. Hyvärinen and Christoph M. Wintersteiger. Parallel satisfiability modulo theories. In *Handbook of Parallel Constraint Reasoning*, pages 141–178. Springer, 2018.

[IGH19] Franc Ivankovic, Dan Gordon, and Patrik Haslum. Planning with global state constraints and state-dependent action costs. In *Proc. of ICAPS*, pages 232–236, 2019.

[IHT+14] Franc Ivankovic, Patrik Haslum, Sylvie Thiébaux, Vikas Shivashankar, and Dana S. Nau. Optimal planning with global numerical state constraints. In *Proc. of ICAPS*, pages 145–153, 2014.

[IM17] Leon Illanes and Sheila A. McIlraith. Numeric planning via abstraction and policy guided search. In *Proc. of IJCAI*, pages 4338–4345, 2017.

[KMS96] Henry A. Kautz, David A. McAllester, and Bart Selman. Encoding plans in propositional logic. In *Proc. of KR*, pages 374–384, 1996.

[KS92] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *Proc. of ECAI*, pages 359–363, 1992.

[KS96] Henry A. Kautz and Bart Selman. Pushing the envelope: planning, propositional logic and stochastic search. In *Proc. of AAAI/IAAI*, pages 1194–1201, 1996.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.

[LÁN+17] Francesco Leofante, Erika Ábrahám, Tim Niemueller, Gerhard Lakemeyer, and Armando Tacchella. On the synthesis of guaranteed-quality plans for robot fleets in logistics scenarios via optimization modulo theories. In *Proc. of IRI*, pages 403–410, 2017.

[LÁN+19] Francesco Leofante, Erika Ábrahám, Tim Niemueller, Gerhard Lakemeyer, and Armando Tacchella. Integrated synthesis and execution of optimal plans for multi-robot systems in logistics. *Information Systems Frontiers*, 21(1):87–107, 2019.

[LÁT18] Francesco Leofante, Erika Ábrahám, and Armando Tacchella. Task planning with OMT: an application to production logistics. In *Proc. of IFM*, pages 316–325, 2018.

[LaV06] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[LD60] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[Leo18a] Francesco Leofante. Guaranteed plans for multi-robot systems via Optimization Modulo Theories. In *Proc. of AAAI*, pages 8020–8021, 2018.

[Leo18b] Francesco Leofante. Optimal multi-robot task planning: from synthesis to execution (and back). In *Proc. of IJCAI*, pages 5771–5772, 2018.

[Leoon] Francesco Leofante. OMTPlan: a tool for optimal planning modulo theories. Under submission.

[LGÁTon] Francesco Leofante, Enrico Giunchiglia, Erika Ábrahám, and Armando Tacchella. Optimal planning modulo theories. Under submission.

[LSHB18] Dongxu Li, Enrico Scala, Patrik Haslum, and Sergiy Bogomolov. Effect-abstraction based relaxation for linear numeric planning. In *Proc. of IJCAI*, pages 4787–4793, 2018.

[LT16] Francesco Leofante and Armando Tacchella. Learning in physical domains: mating safety requirements and costly sampling. In *Proc. of AI*IA*, pages 539–552, 2016.

[LVÁ+16] Francesco Leofante, Simone Vuotto, Erika Ábrahám, Armando Tacchella, and Nils Jansen. Combining static and runtime methods to achieve safe standing-up for humanoid robots. In *Proc. of ISoLA*, pages 496–514, 2016.

[LZ02] Fangzhen Lin and Yuting Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. In *Proc. of AAAI*, pages 112–118, 2002.

[McD00] Drew V. McDermott. The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55, 2000.

[MK99] Amol Dattatraya Mali and Subbarao Kambhampati. On the utility of plan-space (causal) encodings. In *Proc. of AAAI*, pages 557–563, 1999.

[NFL09] Tim Niemueller, Alexander Ferrein, and Gerhard Lakemeyer. A Lua-based behavior engine for controlling the humanoid robot Nao. In *Proc. of RoboCup Symposium*, pages 240–251, 2009.

[Nie08] Ilkka Niemelä. Stable models and difference logic. *Ann. Math. Artif. Intell.*, 53(1-4):313–329, 2008.

[NKVT16] Tim Niemueller, Erez Karpas, Tiago Vaquero, and Eric Timmons. Planning competition for logistics robots in simulation. In *Proc. of PlanRob@ICAPS*, pages 131–134, 2016.

[NLF13] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. Incremental task-level reasoning in a competitive factory automation scenario. In *Proc. of AAAI Spring Symposium*, 2013.

[NLF15] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. The RoboCup Logistics League as a benchmark for planning in robotics. In *Proc. of PlanRob@ICAPS*, pages 63–66, 2015.

[NLLÁ17] Tim Niemueller, Gerhard Lakemeyer, Francesco Leofante, and Erika Ábrahám. Towards CLIPS-based task execution and monitoring with SMT-based decision optimization. In *Proc. of PlanRob@ICAPS*, pages 60–67, 2017.

[NO06] Robert Nieuwenhuis and Albert Oliveras. On SAT modulo theories and optimization problems. In *Proc. of SAT*, pages 156–169, 2006.

[NR16] Alexander Nadel and Vadim Ryvchin. Bit-vector optimization. In *Proc. of TACAS*, pages 851–867, 2016.

[PCCB18a] Chiara Piacentini, Margarita P. Castro, André Augusto Ciré, and J. Christopher Beck. Compiling optimal numeric planning to mixed integer linear programming. In *Proc. of ICAPS*, pages 383–387, 2018.

[PCCB18b] Chiara Piacentini, Margarita P. Castro, André Augusto Ciré, and J. Christopher Beck. Linear and integer programming-based heuristics for cost-optimal numeric planning. In *Proc. of AAAI*, pages 6254–6261, 2018.

[PG86] D.A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[RCL17] RCLL Technical Committee. RoboCup Logistics League – Rules and regulations 2017, 2017. Available at `http://www.robocup-logistics.org/rules`.

[RGPS09] Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. SAT-based parallel planning using a split representation of actions. In *Proc. of ICAPS*, pages 281–288, 2009.

[RHN04] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Parallel encodings of classical planning as satisfiability. In *Proc. of JELIA*, pages 307–319, 2004.

[RHN06] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artif. Intell.*, 170(12-13):1031–1080, 2006.

[Rin09] Jussi Rintanen. Planning and SAT. In *Handbook of Satisfiability*, pages 483–504. IOS Press, 2009.

[Rin11] Jussi Rintanen. Planning with specialized SAT solvers. In *Proc. of AAAI*, pages 1563–1566, 2011.

[Rin12] Jussi Rintanen. Engineering efficient planners with SAT. In *Proc. of ECAI*, pages 684–689, 2012.

[Rin15] Jussi Rintanen. Discretization of temporal models with application to planning with SMT. In *Proc. of AAAI*, pages 3349–3355, 2015.

[Rin17] Jussi Rintanen. Temporal planning with clock-based SMT encodings. In *Proc. of IJCAI*, pages 743–749, 2017.

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition.* Pearson Education, 2010.

[RW10] Silvia Richter and Matthias Westphal. The LAMA planner: guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res.*, 39:127–177, 2010.

[SD05] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a SAT-based planner. *Artif. Intell.*, 166(1-2):194–253, 2005.

[SHMT17] Enrico Scala, Patrik Haslum, Daniele Magazzeni, and Sylvie Thiébaux. Landmarks for numeric planning problems. In *Proc. of IJCAI*, pages 4384–4390, 2017.

[SHT16] Enrico Scala, Patrik Haslum, and Sylvie Thiébaux. Heuristics for numeric planning via subgoaling. In *Proc. of IJCAI*, pages 3228–3234, 2016.

[SHTR16] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. Interval-based relaxation for general numeric planning. In *Proc. of ECAI*, pages 655–663, 2016.

[ST15a] Roberto Sebastiani and Silvia Tomasi. Optimization modulo theories with linear rational costs. *ACM Trans. Comput. Log.*, 16(2):12:1–12:43, 2015.

[ST15b] Roberto Sebastiani and Patrick Trentin. OptiMathSAT: a tool for optimization modulo theories. In *Proc. of CAV*, pages 447–454, 2015.

[ST15c] Roberto Sebastiani and Patrick Trentin. Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions. In *Proc. of TACAS*, pages 335–349, 2015.

[WW99] Steven A. Wolfman and Daniel S. Weld. The LPSAT engine & its application to resource planning. In *Proc. of IJCAI*, pages 310–317, 1999.

[Wyg89] Robert M. Wygant. CLIPS: a powerful development and delivery expert system tool. *Computers & Industrial Engineering*, 17(1–4), 1989.

[xai18] EXplainable AI Planning. In *http://icaps18.icaps-conference.org/xaip/*, 2018.

[ZNL14] Frederik Zwilling, Tim Niemueller, and Gerhard Lakemeyer. Simulation for the RoboCup Logistics League with real-world environment agency and multi-level abstraction. In *Proc. of Robot Soccer World Cup*, pages 220–232, 2014.